PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46 — A1

(11) International Publication Number: **WO 99/49394**

(43) International Publication Date: 30 September 1999 (30.09.99)

(21) International Application Number: PCT/US99/06223

(22) International Filing Date: 22 March 1999 (22.03.99)

(30) Priority Data:
60/078,946 23 March 1998 (23.03.98) US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmond, WA 98052 (US).

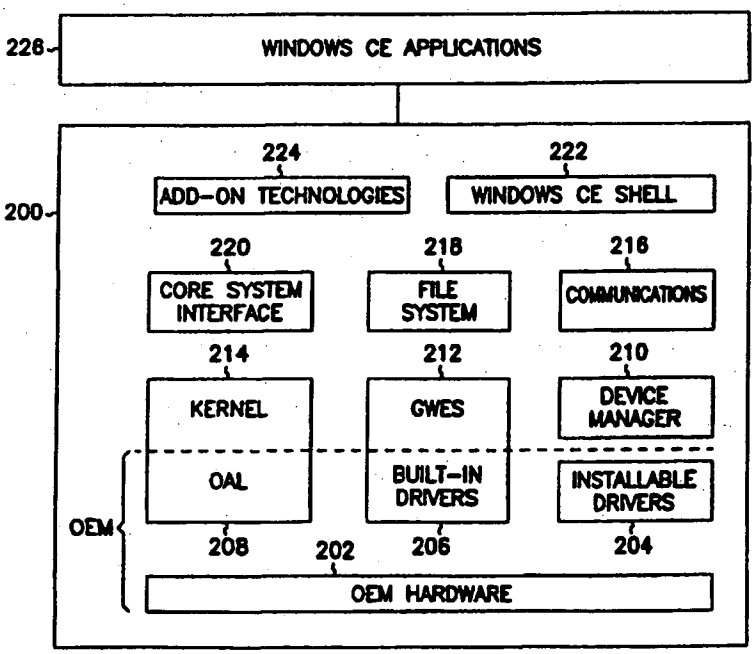
(71)(72) Applicants and Inventors: GAGNE, Rejean [CA/CA]; 4739 Rue Fabre, Montreal, Quebec H2J 3V7 (CA). CAJOLET, Claude [CA/CA]; 535 Avenue Outremont, Outremont, Quebec (CA).

(74) Agent: VIKSNINS, Ann, S.; Schwegman, Lundberg, Woessner & Kluth, P.O. Box 2938, Minneapolis, MN 55402 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: APPLICATION PROGRAM INTERFACES IN AN OPERATING SYSTEM



(57) Abstract

A set of Application Program Interfaces (APIs) for a resource-limited environment are disclosed. The APIs provide a mechanism for a computer application to interface with various components and modules of an operating system for a resource-limited environment. The APIs further provide a mechanism to interface with input/output devices commonly found in embedded systems running in a resource-limited environment.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

APPLICATION PROGRAM INTERFACES IN AN OPERATING SYSTEM

FIELD OF THE INVENTION

This invention relates generally to computer operating systems, and more particularly to application program interfaces for resource limited operating systems.

RELATED FILES

This application claims the benefit of U.S. Provisional Application No. 60/078946, filed March 23, 1998, which is hereby incorporated herein by reference.

COPYRIGHT NOTICE/PERMISSION

BACKGROUND OF THE INVENTION

The rapid evolution of personal computer technology continues to produce personal computers (PCs) that are smaller, cheaper and faster than their predecessors. Where computers once occupied entire rooms, they are now small enough to fit in the palm of a user's hand, hence the name "Palm-size PCs". In addition, PCs are now small enough to be placed in environments outside of the home or office, such as an automobile. Further more, the new PCs may be embedded in a variety of consumer devices and specialized industrial controllers. For the purposes of this application, all of the above-referenced PCs will be referred to collectively as "embedded systems."

The reduced size of embedded systems means that certain sacrifices need to be made. For example, a typical embedded system does not have fixed or removable disk drives such as hard disk, floppy disk, CD-ROM or DVD-ROM drives, with the persistent storage of a typical embedded system comprising flash memory or volatile memory with a battery refresh. In addition, the amount of RAM in the typical embedded system is also limited.

In addition, output resources typical to a desktop PC may be missing or severely limited in an embedded system. For example, the display for a typical embedded system may comprise a small LCD screen with limited resolution and capable of displaying only grayscale or a limited number of colors. In certain environments, such as the automobile, the display may be an LCD screen with a limited number of fixed icons and text areas. The display may be augmented with a computerized speech facility.

Similarly, input resources may be limited or adapted for use in embedded systems. For example, many embedded systems do not have a mouse or other pointing device. In addition, some hand-held devices do not have a physical keyboard. Such embedded devices may use a touch sensitive display in conjunction with a virtual keyboard placed on the display. In addition, embedded devices may employ speech recognition for input.

As a result of the above, specialized operating systems capable of running in the resource-limited environment of the embedded system have been developed. An example of such an operating system is the Windows CE™ operating system from Microsoft Corporation.

Applications running on the embedded system must also be capable of running in the resource limited environment described above. In embedded systems comprising Palm-size PCs, these applications are typically specialized versions of applications available on the bigger siblings of the Palm-size PC, such as calendar programs, personal information managers, calculators, dictionaries and the like.

In other environments, the applications running on the embedded system may be more specialized. For example, in an AutoPC, the applications may comprise applications that interface with an audio system, applications that report and use position and navigation information, and applications that monitor the condition and state of various other systems present in the automobile.

In order to accommodate a large number of different application needs, operating systems typically provide APIs (Application Programming Interfaces) to a wide variety of functionality that is common to many differing applications. Any one application generally uses only a small subset of the available APIs. Providing a wide variety of APIs frees application developers from having to write code that would have to be potentially duplicated in each application. However, in the resource limited environment of the embedded system, there is typically a much more limited set of APIs available. This is because there is generally insufficient persistent and non-persistent memory available to support a large number of different APIs. Thus, a developer writing an application for an embedded system may find that he or she must develop code that would ordinarily be provided by the operating system in a desktop's or other larger computer's operating system.

As a result of the above, there is a need in the art for an operating system capable of running in the resource limited environment of an embedded system. Such an operating system should be customizable and adaptable to the wide variety environments that system designers may choose to place embedded systems, allowing developers to include only those components and modules that are necessary for a particular environment. In addition, the operating system should include APIs to operating system provided components in order prevent applications designers from having to duplicate commonly needed code. Finally, the operating system should provide APIs for components and modules that meet the unique input and output needs of an embedded system.

SUMMARY OF THE INVENTION

The above-mentioned shortcomings, disadvantages and problems are addressed by the present invention, which will be understood by reading and studying the following specification.

A system is presented that includes a set of Application Program Interfaces (APIs) for a number of software modules and components for resource limited environments. One example of a resource limited environment is the embedded system, which comprises a variety of consumer devices and specialized industrial controllers, along with hand-held, or palm-size personal computers.

One aspect of the system is that the combination of components and modules included in an operating system for resource limited environments is customizable and flexible. This allows an embedded system designer to include only those components and modules that are necessary for a particular environment. As a result, scarce memory is not consumed by unneeded components, allowing more memory to be devoted to applications and other modules and components that are needed in the embedded system.

Another aspect of the system is that APIs are provided that meet the unique input and output needs of the typical embedded system. For example, many embedded systems do not provided a keyboard or mouse for input. The system provides APIs to components and modules that provide alternative mechanisms of providing input. These alternative mechanisms include APIs to handwriting recognition engines that "read" strokes on a touch sensitive screen, and APIs to voice input components that allow a user to issue spoken commands to the system. Further, the system provides APIs to components that output audible speech for those environments where a display monitor is impractical.

Another aspect of the system is that the handling of "out of memory" conditions is customizable by an embedded system designer. This is important to systems with limited resources, because out of memory conditions are more likely to occur.

A further aspect of the system is that an API to a position and navigation component is provided. This is useful for embedded system environments that are mobile, such as automobiles, trucks, and boats.

The APIs summarized above, and various other APIs, will be described in detail in the sections that follow.

The present invention describes systems, clients, servers, methods, and computer-readable media of varying scope. In addition to the aspects and advantages of the present invention described in this summary, further aspects and advantages of the invention will become apparent by reference to the drawings and by reading the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced;

FIG. 2 is a diagram illustrating a system-level overview of exemplary embodiments of an operating system for a resource limited environment; and

FIG. 3 is a diagram further illustrating the relationship of modules, components and APIs according to an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, electrical and other changes may be made without

departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

The detailed description is divided into four sections. In the first section, the hardware and the operating environment in conjunction with which embodiments of the invention may be practiced are described. In the second section, a system level overview of the invention is presented. In the third section, various APIs are presented allowing applications to interface with various modules and components of an operating system. Finally, in the fourth section, a conclusion of the detailed description is provided.

Hardware and Operating Environment

FIG. 1 is a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced. The description of FIG. 1 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer, a hand-held or palm-size computer, or an embedded system such as a computer in a consumer device or specialized industrial controller. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCS, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing

environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

The exemplary hardware and operating environment of FIG. 1 for implementing the invention includes a general purpose computing device in the form of a computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that operatively couples various system components including the system memory to the processing unit 21. There may be only one or there may be more than one processing unit 21, such that the processor of computer 20 comprises a single central-processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment. The computer 20 may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited.

The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory may also be referred to as simply the memory, and includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within the computer 20, such as during start-up, is stored in ROM 24. In one embodiment of the invention, the computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. In alternative embodiments of the invention, the functionality provided by the hard disk drive 27, magnetic disk 29 and optical disk drive 30 is emulated using volatile or non-volatile RAM in order to conserve power and reduce the size of the system. In these alternative

embodiments, the RAM may be fixed in the computer system, or it may be a removable RAM device, such as a Compact Flash memory card.

In an embodiment of the invention, the hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for the computer 20. It should be appreciated by those skilled in the art that any type of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24, or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, touch sensitive pad, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). In addition, input to the system may be provided by a microphone to receive audio input.

A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In one embodiment of the invention, the monitor comprises a Liquid Crystal Display (LCD). In

addition to the monitor, computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. These logical connections are achieved by a communication device coupled to or a part of the computer 20; the invention is not limited to a particular type of communications device. The remote computer 49 may be another computer, a server, a router, a network PC, a client, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local-area network (LAN) 51 and a wide-area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN-networking environment, the computer 20 is connected to the local network 51 through a network interface or adapter 53, which is one type of communications device. When used in a WAN-networking environment, the computer 20 typically includes a modem 54, a type of communications device, or any other type of communications device for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It is appreciated that the network connections shown are exemplary and other means of and communications devices for establishing a communications link between the computers may be used.

The hardware and operating environment in conjunction with which embodiments of the invention may be practiced has been described. The

computer in conjunction with which embodiments of the invention may be practiced may be a conventional computer an hand-held or palm-size computer, a computer in an embedded system, a distributed computer, or any other type of computer; the invention is not so limited. Such a computer typically includes one or more processing units as its processor, and a computer-readable medium such as a memory. The computer may also include a communications device such as a network adapter or a modem, so that it is able to communicatively couple other computers.

System Level Overview

A system level overview of the operation of an exemplary embodiment of the invention is described by reference to FIGs. 2 and 3. The concepts of the invention are described as operating in a multiprocessing, multithreaded operating environment on a computer, such as computer 20 in FIG. 1. The exemplary operating environment comprises what is known in the art as an operating system. In this environment one or more applications, such application 226, interface with various modules and components of the operating system. In addition, the various modules and components of the operating system interface with each other. Finally, the modules, components and applications interface with hardware 202 present on the computer through what is known in the art as a device driver module, and through an Original Equipment Manufacturer (OEM) adaptation layer 208. In one embodiment of the invention, there are two types of device drivers, built-in drivers 206 and installable drivers 204. The various modules will now be described in further detail.

The core system interface 220 is the module through which applications can access the operating system. The core system interface 220 includes functions to transfer API calls to the appropriate operating system server process.

In addition to including or exporting the APIs selected, the core system interface 220 includes components to support the following:

- Localization
- Local heap and memory allocation
- Serial port device driver thunks
- Telephony API (TAPI)

The shell module 222 manages the user interface and handles such tasks as launching software applications. In one embodiment of the invention, the operating system provides shell components that enable an embedded system designer to develop a customized shell 222 that satisfies the requirements of the target platform. Included in these components are:

- A Control Panel with applets familiar to desktop Windows users. The following applets are included: Communications; Display; Keyboard; Network; Owner; Password; Power; Regional Settings, Remove Programs; Pointing Device Settings (Stylus); Sounds and Volume.
- A Notification API that lets an application register its name and an event with the system. When the event occurs, the kernel will automatically start the named application. The API also allows an application to register a specific date and time at which the application should start.
- Common controls and common dialogs, which are designed to provide to the user clear, simple, and meaningful information and a means to furnish input to the system and applications as needed.
- A command line processor (that is, a console application) that supports a set of standard input and output API calls.
- Connectivity components (for example, to support remote application programming calls) between the development workstation and the embedded system target platform.

In conjunction with a desktop, the shell module 222 also includes a desktop and task manager component that can be optionally included or

replaced. The task manager component includes the following basic functionality:

- An Active Tasks list of all the currently running, top-level applications;
- A Run button that allows a user to launch a software application;
- A Switch To button that allows a user to switch to an application selected in the Active Tasks listbox.
- An End Task button that allows a user to terminate an application selected in the Active Tasks listbox.
- A Cancel button that allows a user to close the Task-Manager window.
- Monitors the level of main battery and backup battery power (for battery-operated target platforms) and displays an appropriate warning dialog box.
- Monitors system memory usage in the system and sends a message to all top-level windows when the available system memory drops below a specific threshold. This allows applications to respond to the message by reducing their memory usage as much as possible.

The Add-on Technologies module 224 allows an embedded system developer to optionally include components such as OLE/COM automation that supports development of ActiveX-based applications, an active desktop shell and an Internet browser. Other components that can be included are Visual Basic run-time and Java script, and a subset of the Microsoft Foundation Classes (MFC). A further optional component that can be provided is a handwriting recognition engine with associated APIs. In one embodiment of the invention, handwriting applications interface with a touch sensitive input device through a component providing a software interface to the touch sensitive device.

The kernel module 214 represents the base operating system functionality that must be present on all platforms. The kernel module includes memory management, process management, exception handling, and support for multitasking and multithreading.

In one embodiment of the invention, the kernel 214 is designed specifically for small, fast, embedded devices. In this embodiment, the kernel supports a single 4GB address space (a 2GB virtual address and a 2GB physical address range). In an embodiment of the invention, this 4GB address space is divided into 33 "slots", each of which has a size of 32MB. The kernel protects each process by assigning each process to a unique, open slot in memory. The invention, however, is not limited to any particular physical or virtual address space or slot size, and other sized may be chosen as those of skill in the art will recognize.

The kernel 214 protects applications from accessing memory outside of their allocated slot by generating an exception. Applications can check for and handle such exceptions by using the try and except Windows CE functions. In one embodiment of the invention, the system is limited to 32 processes, but the number of threads running in a process is limited only by the amount of available memory. Those of skill in the art will appreciate that other values for the maximum number of processes could be chosen.

The file system module 218 contains the functions that support persistent storage on the embedded system target platform. This storage is referred to as the "object store" and includes three different ways to store user data:

- The file system. The file system typically supports common file manipulation functions, such as functions to create files and directories, read and write to files, and retrieve file and directory information.

- The registry. The system registry is similar to the registries of the Windows 95 and Windows NT operating systems. The registry for all applications, including the applications bundled in ROM, is stored in the object store.

- The Database API. The operating system, in one embodiment of the invention, has its own structured storage to offer an alternative to exposing user and application data in files or the registry. For example, a database is useful for storing raw data that an application will process before displaying to the end-user. Hand-held PC applications typically store schedule and contact information in databases.

In one embodiment of the invention, the file system managed by file system module 218 is a transactioned system to reduce the possibility that data will be lost due to a critical failure, such as loss of power. Additionally, in one embodiment of the invention, the file system module 218 implements a scheme (transactioned) of "mirroring" to mirror or track file system operations (not transactioned). The purpose for this implementation is to be able to restore a file system volume in the case that power is lost during a critical sequence of operations being performed on the volume.

In one embodiment of the invention, the operating environment combines the Win32 User and GDI (Graphics Device Interface) libraries into a GWES (Graphics, Windowing, and Events Subsystem) module 212. The event manager and window manager are analogous to Win32 User, and the Win32 GDI is replaced with a smaller GDI more suitable to embedded systems. The GWES module 212 includes multiplatform GDI components (supporting an associated display driver) that support color and grayscale display, palette management, TrueType fonts, Raster fonts, cursors, and printer device contexts (DCs).

The GWES module 212 also supports a window management component that provides API functions tailored for the smaller display sizes typical of embedded operating systems.

The operating environment of various embodiments of the invention is event-driven. GWES module includes components to handle events, which in one embodiment of the invention are implemented as messages.

Communications module 216 includes a variety of communications component options to support communications hardware. This includes serial, parallel, and network (wired and wireless) communications. Communications module 216 includes the following selectable communications features:

- Serial I/O support
- Networking support including:
  - NDIS 4.0 for local area networking
  - PPP and SLIP for serial link and modem networking
  - Client-side Remote Access Server (RAS)
  - Internet protocols
  - Telephony API (TAPI)
  - PC Card support
  - Infrared transceiver support

In one embodiment of the invention, an embedded systems designer must develop the OEM adaptation layer 208 to create the platform specific kernel module 214. The OEM Adaptation Layer (OAL) module 208 allows an embedded system developer to adapt the operating system for a specific target platform by creating a thin layer of code that resides between the kernel module 214 and the target platform hardware 202. The OAL module 208 is specific for a particular CPU and target platform.

The OAL module 208 includes interfaces such as the following:

- Interrupt service routine (ISR) handlers to support device drivers
- Real-time clock (RTC)
- Interval timer (used for the scheduler operation)

In one embodiment of the invention, the RTC and interval timer does not need to be adapted because it is provided on the CPU. In this case, these interfaces are implemented in the kernel module 214 rather than in the OAL 208.

In addition to managing such functions as timing and power, the primary purpose of the OAL is to expose the target platform's hardware 202 to the kernel module 214. That is, each hardware interrupt request line (IRQ) is associated with one interrupt service routine (ISR). When interrupts are enabled and an interrupt occurs, the kernel calls the registered ISR for that interrupt.

Built in drivers 206 are device drivers that are linked with GWES module 212 when building the operating system. Examples of such drivers are the notification LED driver or the battery driver. These drivers are called "built-in device drivers" because they ultimately form part of the same executable image as the rest of the operating system. Built-in device drivers each have a custom interface to the rest of operating system.

Device Manager module 210 is a module that handles installable device drivers. In one embodiment of the invention, The Device Manager 210 performs the following tasks:

- Initiates the loading of a driver at system start up, or when it receives a notification that a third-party peripheral has been attached to the target platform. For example, when a PC Card is inserted, Device Manager 210 will attempt to locate and load a device driver for that PC Card.
- Registers special filesystem entries with the kernel that map the Stream I/O Interface functions used by applications to the implementation of those functions in an installable device driver.
- Finds the appropriate device driver by obtaining a Plug and Play ID or by invoking a detection routine to find a driver that can handle the device.
- Loads and tracks drivers by reading and writing registry values.
- Unloads drivers when their devices are no longer needed. For example, Device Manager 210 will unload a PC Card device driver when the card is removed.

In one embodiment of the invention, Installable Device Drivers 204 exist as standalone DLLs (Dynamic Link Library) that are managed by the Device

Manager 210. Installable device drivers 204 support some types of native devices, any peripheral devices that can be connected to the target platform, and any special purpose devices that are added to the platform. This covers devices such as modems, printers, digital cameras, PC Cards (also known as PCMCIA cards), and others.

In one embodiment of the invention, installable device drivers 204 use a common interface by which their services are exposed to applications. This interface is the Stream I/O Interface.

A description of the relationships between components, modules and the APIs they expose to applications is presented with reference to FIG. 3. A module 308 is a major functional block of an operating environment such as operating system 200 of FIG. 2. Module 308 exposes an API 302 to applications such as application 226 of FIG. 2 that allows the application to interface and call methods or functions implemented by the module 308.

Modules may optionally include one or more components 306. Components 306 are groups of functions and data that provide capabilities on a smaller scale than modules 308. Like a module 308, a component 306 also exposes an API 304 that other applications, modules, and components may use to call methods or functions implemented by the component 306.

As can be seen from the discussion above, the various embodiments of the invention provide advantages over prior systems. One benefit is that the operating system is modular. This allows an embedded system designer to create an operating environment that is optimized for their unique hardware development platform and application. The developer can select varying combinations of the above-described modules and components for inclusion in the operating environment. For example, a developer can build an embedded operating system that contains the kernel and a selected set of communications but does not provide a graphical user interface. Thus, the invention is not limited to any particular combination of modules and components.

The various embodiments of the invention also provides a mechanism for developers to conserve the limited memory resources of a typical embedded system, because only those modules and components having APIs that are necessary for the operating environment need be included.

APIs in a Resource Limited System

The previous section presented a system level overview of modules and components included in a typical operating system for a system with limited resources. This section, along with the sub-sections that follow, present novel APIs and data structures related to the modules and components described above. The APIs detailed below are described in terms of the C/C++ programming language. However, the invention is not so limited, and the APIs may be defined and implemented in any programming language, as those of skill in the art will recognize. Furthermore, the names given to the API functions and parameters are meant to be descriptive of their function, however other names or identifiers could be associated with the functions and parameters, as will be apparent to those of skill in the art. Six sets of APIs and data structures will be presented: Handwriting Recognition APIs, Position and Navigation APIs, Speech related APIs, Out of Memory APIs, Database APIs and Active Synch Data Structures.

1. Handwriting Recognition APIs

A handwriting recognition component is available in the Add-On Technologies module 224 (FIG. 2). The handwriting recognition component implements a handwriting recognition engine. In one embodiment of the invention, the engine receives "ink" in the form of a plurality of strokes on a touch sensitive screen. The strokes are then sent from applications to the engine using a variety of APIs. The engine then attempts to interpret the strokes as alphanumeric characters. The interpreted characters are returned to the application via an API. In one embodiment of the invention, the characters are

interpreted as English language characters. In alternative embodiments of the invention, the characters are interpreted in other languages.

The handwriting recognition component is particularly useful in embedded systems that have a touch sensitive display, but no keyboard. Applications that require alphanumeric input can use the characters received from the engine as if they had been typed at a keyboard.

Further details on the APIs used by applications that interface with a handwriting recognition engine are presented in the sub-section entitled "Detailed Description of a Handwriting Recognition API."

2. Position and Navigation APIs and Data Structures

A Position and Navigation component is available in the Add-On Technologies module. The Position and Navigation component allows an application to interface with a positioning device (also referred to as a positioning and navigation device) such as an Apollo GPS system. Such an interface is useful when the embedded system is located in a mobile article such as an automobile or truck. In one embodiment of the invention, the embedded system is the AutoPC.

Further details on the APIs for the Position and Navigation module are found in the sub-section entitled "Detailed Description of a Position and Navigation API." Also, further details on data structures used by the Position and Navigation Module and related APIs are found in the sub-section entitled "Detailed Description of Data Structures for a Position and Navigation System."

3. Speech Related APIs

The Add-On Technologies module contains several speech-related components that expose APIs for application use. These components include a text-to-speech component, a voice-to-text component, and a voice command component. In general, these components are intended for environments where

input and output devices are limited, and where a user's interaction with the embedded system is via speech. An example of such an environment is the AutoPC. Because the driver must use their hands in the operation of the automobile, interaction with the AutoPC is via a speech interface, where input commands are spoken by the user, and output from the PC is converted from text to speech.

Further details on the text-to-speech APIs are presented in the sub-section entitled "Detailed Description of a Speech-to-Text API." Further details on the voice command and speech to text APIs are presented in the sub-sections entitled "Detailed Description of a Voice Command API", "Detailed Description of Data Structures for a Voice Command API, and "Detailed Description of a Voice Command API for an AutoPC."

4. Out of Memory API

The Out of Memory API is a component of the GWES module. This component allows an embedded system developer to replace the default action that occurs when the operating system detects that the system is running out of available memory in which to run applications or place data.

The Out of Memory component is significant to an operating system intended for limited resource environments, because the condition is more likely to occur in an embedded system than in a desk-top system. The API exposed provides a standardized way for the operating system to call customized software that meets the specific needs of an embedded system developer.

Further details on the out of memory API are presented in the sub-section entitled "Detailed Description of an Out-of-Memory API."

5. Database API

As discussed above in reference to FIG. 2, the file system module 218 may optionally include a database component. The database component allows

applications to create and maintain databases as file system objects. Applications make calls to various API functions that maintain the database. These functions include functions that create new databases, open existing database, delete databases, seeks particular records in databases, read records from databases and write records to databases. In addition, the Database API includes functions that navigate through a list of databases of a given type. Further details regarding the Database API are presented in the sub-section entitled "Detailed Description of a Database API."

6. ActiveSync Data Structures

ActiveSync is a component available in the Add-On Technologies module. The ActiveSync component provides a service that allows applications to compare two objects to determine if one of the objects needs to be updated in order for the objects to be "synchronized", that is, the same. Typically the objects are file system objects containing application data. ActiveSync is particularly useful when applied to hand-held PCs. This is because the user often will update data maintained in a file system object on the hand-held PC, and then need to update a file on a desk-top PC so that the two files contain the same data. For example, hand-held PCs typically provide an application such as a Personal Information Manager that maintains a database of information, including telephone numbers. If a user maintains a similar database of telephone numbers on both their hand-held PC and their desk-top PC, it is desirable that the two telephone directories reflect updates made to either the hand-held PC or desk-top PC database. ActiveSync allows a user to accomplish this.

In one embodiment of the invention, several data structures are employed that enable ActiveSync to correctly compare and perform updates to corresponding objects. The first data structure is the CONFINFO data structure. This data structure is used to retrieve information about two potentially conflicting items. In one embodiment of the invention, an ActiveSync Server

presents the information in the CONFINFO data structure to a user via a dialogue box to allow the user to choose an option for resolving the conflict. Further details regarding the CONFINFO data structure are presented in the sub-section entitled "Detailed Description of Data Structures for a Synchronization API."

A second data structure used by the Active Synch component is the OBJNOTIFY structure. The OBJNOTIFY data structure is used to notify the ActiveSync service provider that an object in the file system has changed or been deleted. Further details regarding the OBJNOTIFY data structure are presented in the sub-section entitled "Detailed Description of Data Structures for a Synchronization API."

Detailed Description of Data Structures for a Synchronization API

Chapter 106

HREPLITEM

The HREPLITEM structure is used as a handle to a data object stored by a client. It is used as a generic handle to refer to either HREPLOBJ or HREPLFLD.

Syntax: typedef struct_REPLITEM FAR *HREPLITEM;

At a Glance:

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members:

HREPLFLD
Handle to a data object stored by a client.

HREPLFLD

The HREPLFLD structure is used as a handle to a folder stored by a client.

Syntax: typedef struct_REPLFLD FAR *HREPLFLD;

At a Glance:

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members:

HREPLFLD
Handle to a folder stored by a client.

HREPLOBJ

The HREPLOBJ structure is used as a handle to an object stored by a client.

Syntax: typedef struct_REPLOBJ FAR *HREPLOBJ;

At a Glance:

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members:

HREPLITEM
Handle to an object stored by clients.

CONFINFO

The CONFINFO structure is used to retrieve information about two conflicting items. The server presents this information to the user via a dialog box so the user can choose an option for resolving the conflict.

Syntax

```
typedef struct tagConflnfo{
    UINT        cbStruct;
    HREPLFLD    hFolder;
    HREPLITEM   hLocalItem;
    HREPLITEM   hRemoteItem;
    char        szLocalName[MAX_OBJTYPE_NAME];
    char        szLocalDesc[512];
    char        szRemoteName[MAX_OBJTYPE_NAME];
    char        szRemoteDesc[512];
} CONFINFO, *PCONFINFO;
```

At a Glance

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members

cbStruct
: Size of this structure.

hFolder
: Handle representing the folder where the objects are stored.

hLocalItem
: Handle representing the local object.

hRemoteItem
: Handle representing the remote object.

szLocalName
: Name of the local object client would like to show to the user.

szLocalDesc
: Description of the local object client would like to show to the user.

szRemoteName
: Name of the remote object client would like to show to the user.

szRemoteDesc
: Description of the remote object client would like to show to the user.

See Also IReplStore::GetConflictInfo

OBJNOTIFY

The OBJNOTIFY structure is used to notify the ActiveSync service provider that an object in the Windows CE file system has changed or been deleted.

```
typedef struct tagObjNotify{
    UINT            cbStruct;
    OBJTYPENAME     szOBJType[MAX_OBJTYPE_NAME];
    UINT            uFlags;
    UINT            uPartnerBit;
    CEOID           oidObject;
    CEOIDINFO       oidInfo;
    UINT            cOidChg;
    UINT            cOidDel;
    UINT            *poid
} OBJNOTIFY, *POBJNOTIFY;
```

At a Glance

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members

cbStruct
: Input. Size of the structure in bytes.

SzObjType
: Input, the object type name.

uFlags
: Input Flags.

ONF_FILE
: the object is a file.

ONF_DIRECTORY
: the object is a directory.

ONF_DATABASE
: the object is a database.

ONF_RECORD
: the object is a record.

ONF_CHANGED
: set if the file system object is changed.

ONF_DELETED
: set if the file system object is deleted.

ONF_CLEAR_CHANGE
: client should clear the change bit for the object whose object identifier is pointed at by *poid*.

ONF_CALL_BACK
: output. Client asks server to call ObjectNotify two seconds later.

ONF_CALLING_BACK
: set if this call is a result of ONF_CALL_BACK being set earlier.

uPartnerBit
Input. It is 1 if the desktop currently connected is partner #1, and it is 2 if the desktop is partner #2.

oidObject
Input. This is the OID of the file system object, representing a file, a database, or a database record.

OidInfo
Input. Stores information about the object (if the object has not been deleted).

cOidChg
Output. When ONF_CHANGED is set, this is the number of oid's that should be replicated. Set to 0 if no object should be replicated because of this change.

When both ONF_CHANGED and ONF_DELETED are not set, this is the number of oid's in the first part of the list for objects that are changed.

cOidDel
Output. When ONF_DELETED is set, this is the number of deleted oids that should be replicated. Set to 0 if no object should be replicated because of this delete.

When both ONF_CHANGED and ONF_DELETED are not set, this is the number of oids in the later part of the list for objects that are not changed.

poid
Output. Points to an array of oid's that should be marked as needs to be replicated first cOidChg elements are for the changed objects, the last cOidDel elements are for the deleted objects Note that, memory pointed to by this pointer is owned by the ActiveSync service provider. It will not be freed by replication.

Remarks

This structure is passed to the ObjectNotify function to inform the provider that an event that changes or deletes an object in the Windows CE file system has occurred. The provider should return, via this structure, how many replication objects have changed or been deleted because of this change or deletion to a file system object.

When ONF_CHANGED is set, *cOidChg* is the number of object id's in the list that should be synchronized (*cOidDel* is not used).

When ONF_DELETED is set, *cOidDel* is the number of deleted object id's in the list that should by synchronized (*cOidChg* is not used).

See Also

ObjectNotify

OBJUIDATA

The OBJUIDATA structure is used by IReplStore::GetObjTypeUIData to send UI related data about an object type to the Store.

Syntax

```
typedef struct tagObjUIData{
    UINT    cbStruct;
    HICON   hIconLarge;
    HICON   hIconSmall;
    char    szName[MAX_PATH];
    char    szSyncText[MAX_PATH];
    char    szTypeText[80];
    char    szP1TypeText[80];
} OBJUIDATA, *POBJUIDATA;
```

At a Glance

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members

cbStruct
: The size of this structure.

hIconLarge
: The handle of a large icon used in the list view display of the H/PC Explorer.

hIconSmall
: The handle of a small icon used in the list view display of the H/PC Explorer.

szName
: Text displayed in the "Name" column of the H/PC Explorer.

szSyncText
: Text displayed in the "Sync Copy In" column of the H/PC Explorer.

szTypeText
: Text displayed in the "Type" column of the H/PC Explorer.

szP1TypeText
: Plural form of text displayed in the "Type" column of the H/PC Explorer.

See Also

IReplStore::GetObjTypeUIData

REPLSETUP

The REPLSETUP structure is used to initiate the object handler.

Syntax

```
typedef struct tagReplSetup{
        UINT        cbStruct;
        BOOL        fRead;
        DWORD       dwFlags;
        HRESULT     hr;
        TCHAR       szObjType[MAX_OBJTYPE_NAME];
        IReplNotify *pNotify;
        DWORD       oid;
        DWORD       oidNew;
        IReplStore  *pStore;
        HREPLFLD    hFolder;
        HREPLITEM   hItem;
} REPLSETUP, *PREPLSETUP;
```

At a Glance

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members

cbStruct
Input. Size of this structure.

fRead
Input. TRUE if setting up for reading (serializing) the object. FALSE if setting up for writing (deserializing) the object.

dwFlags
Reserved by replication.

Hr
Output. Result of the read/write operation.

szObjType
Input. Name of the object type.

pNotify
Input. Pointer to IReplNotify::IUnknown interface.

Oid
Input. Object ID of the object.

oidNew
Output. Object ID of the new object. This is different from the oid if a new object was created during writing.

pStore
Input. Exists in desktop only. Points to IReplStore interface. This is unused for device side use.

hFolder
Input. Exists in desktop only. Handle of the folder. This is unused for device side use.

hItem

Input or Output. Exists in desktop only. Handle of the object to be read or written. This is unused for device side use.

See Also IReplObjHandler::Setup

STOREINFO

The STOREINFO structure is used to identify an instance of the store.

Syntax

```
typedef struct tagStoreInfo {
    UINT    cbStruct;
    UINT    uFlags;
    TCHAR   szProgId[256];
    TCHAR   szStoreDesc[200];
    UINT    uTimerRes;
    UINT    cbMaxStoreId;
    UINT    cbStoreId;
    LPBYTE  lpbStoreId;
} STOREINFO, *PSTOREINFO;
```

At a Glance

Header file:	cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Members

cbStruct

Size of this structure.

uFlags

Output. Combination of the following flags:

SCF_SINGLE_THREAD

Set if the implementation only supports single thread operation.

SCF_SIMULATE_RTS

Set if the implementation wants to simulate detection of real-time change/deletes.

szProgId

Output. ProgID name of the store object.

szStoreDesc

Output. Description of the store, will be displayed to the user.

uTimerRes

Input/Output. Resolution of timer in microseconds. 5000 by default. Applicable only when SCF_SIMULATE_RTS is set in *uFlags*.

cbMaxStoreId
Input. Max. size of the store ID that can be stored in buffer pointed by IpbStoreId.
cbStoreId
Output. Actual size of the store ID stored in buffer pointed by *lpbStoreId*.
IpbStoreId
Output pointer to a buffer of anything that uniquely identifies the current store instance, for example, a schedule file.

Remarks — Note that calls to the IReplStore interface methods can come from different threads. If the client does not support multi-threading, it must set *fSingleThreadOnly* to FALSE, so the server will serialize the calls to the methods and make them all come from the primary thread of the application. *szStoreDesc* can have a value such as "Schedule+File". It is displayed to the user whenever the store ID indicates a different store, such as a different Schedule+file, has been installed.

See Also — IReplStore::GetStoreInfo

DEVINFO

The DEVINFO structure is used to store information about a device.

```
typedef struct tagDevInfo {
        DWORD       pid;
        char        szName[MAX_PATH];
        char        szType[80];
        char        szPath[MAX_PATH]
} DEVINFO, *PDEVINFO;
```

At a Glance — Header file:
Platforms:
Windows CE versions:

Members

pid
Device identifier.
szName
Device name.
szType
Device type.
szPath
Device path.

OBJTYPEINFO

The OBJTYPEINFO structure is used to store information about an object type.

```
typedef struct tagOBJTypeInfo {
        UINT            cbStruct;
        OBJTYPENAMEW    szObjType;
        UINT            uFlags;
        WCHAR           szName[80]
        UINT            cObjects;
        UNIT            cbAllObj;
        FILETIME        ftLastModified
} OBJTYPEINFO, *POBJTYPEINFO;
```

At a Glance

Header file:
Platforms:
Windows CE versions:

Members

cbStruct
Input. The size of the structure in bytes.

szObjtype
Input. The object type name.

uFlags
Reserved.

szName
Output. The name of a file system object storing all these objects.

cObjects
Output. The number of existing objects of this type.

cbAllObj
Output. The total number of bytes used to store existing objects.

ftLastModified
Output. The last time any object was modified.

Detailed Description of a Synchronization API

Chapter 8

IReplNotify : IUnknown

An ActiveSync service manager implements the IReplNotify:Notify interface, which can be used by an ActiveSync service provider to notify the ActiveSync service manager of certain events taking place in the ActiveSync service provider's store.

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Methods	Description
IReplNotify::GetWindow	Obtains a handle to the window that must be used as a parent for any modal dialog or message box that an ActiveSync service provider wants to display.
IReplNotify::OnItemCompleted	Used internally by the ActiveSync service manager. An ActiveSync service provider should not call this explicitly.
IReplNotify::OnItemNotify	Notifies the ActiveSync service manager that an item has been created, deleted, or modified.
IReplNotify::QueryDevice	Used to ask for information about a device.
IReplNotify::SetStatusText	Sets the text to be displayed on the Explorer Window status control.
IUnknown::AddRef	Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a specified object.
IUnknown::QueryInterface	Returns a pointer to a specified interface on an object to which a client currently holds an interface pointer. This method must call IUnknown::AddRef on the pointer it returns.

IUnknown::Release	Decrements the reference count for the calling interface on an object. If the reference count on the object falls to 0, the object is freed from memory.

Remarks — The IReplNotify : IUnknown interface is implemented and exposed by the ActiveSync service manager. If the store is capable of detecting changes and deletions to the objects as they occur, an ActiveSync service provider should use the interface to notify the ActiveSync service manager of these changes and deletions. This is more efficient than enumerating the changes and comparing time stamps.

IReplNotify::GetWindow

The IReplNotify::GetWindow method obtains a handle to the window that must be used as a parent for any modal dialog or message box that an ActiveSync service provider wants to display.

Syntax

```
HRESULT GetWindow(
UINT uFlags
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

uFlags
Reserved; always 0.

See Also — IReplNotify

IReplNotify::OnItemCompleted

The IReplNotify::OnItemCompleted method is used internally by the ActiveSync service manager. An ActiveSync service provider should never call this method explicitly.

Syntax

```
HRESULT OnObjectCompleted(
PREPLSETUP pSetup
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
pSetup
Pointer to a REPLSETUP structure.

See Also
IReplNotify

IReplNotify::OnItemNotify

The IReplNotify::OnItemNotify method notifies the ActiveSync service manager that an object has been created, deleted, or modified.

Syntax

```
HRESULT OnItemNotify(
UINT uCode,
LPSTR lpszProgId,
LPSTR lpszObjType,
HREPLITEM hItem,
ULONG ulFlags
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
uCode
Code that describes what happened. Possible values include the following:
RNC_CREATED
Object was created.
RNC_MODIFIED
Object was modified.
RNC_DELETED
Object was deleted.
RNC_SHUTDOWN
The store has been shut down. Windows CE Services should unload the module immediately.
lpszProgId
Programmatic identifier of the store.
IpszObjType
Name of the object type.
hItem
Handle of the concerned item.
ulFlags
Reserved.

Remarks — If the store is capable of detecting changes and deletions as they occur, an ActiveSync service provider should call the IReplNotify::OnItemNotify method immediately after any changes or deletions are detected.

See Also — IReplNotify

IReplNotify::QueryDevice

The IReplNotify::QueryDevice method is used to ask for information about a device.

Syntax

```
void QueryDevice(
UINT uCode,
LPVOID *ppvData
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

uCode

Input parameter. Possible values include the following:

QDC_SEL_DEVICE
: Requests information for the selected device. In this case, **ppvData* points to the DEVINFO structure containing the information for the device.

QDC_CON_DEVICE
: Requests information for the connected device. In this case, **ppvData* points to the DEVINFO structure containing the information for the device.

QDC_SEL_DEVICE_KEY
: Gets a registry key that can be used to store selected device-specific settings. In this case, **ppvData* points to HKEY. The caller must close the registry key when its usage is over.

QDC_CON_DEVICE_KEY
: Gets a registry key that can be used to store connected device-specific settings. In this case, **ppvData* points to HKEY. The caller must close the registry key when its usage is over.

ppvData

Output parameter. Depending on *uCode*, this can point either to a DEVINFO structure or HKEY.

IReplNotify::SetStatusText

The IReplNotify::SetStatusText method sets the text to be displayed on the Explorer Window status control.

Syntax

```
HRESULT SetStatusText (
LPSTR lpszText
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

lpszText

Pointer to a status text string.

Remarks

Status messages should be advisory only. Use modal dialog boxes or message boxes for information that requires user intervention.

See Also

IReplNotify

IReplObjHandler : IUnknown

The IReplObjHandler : IUnknown interface implements all required functions related to the serialization and deserialization of an object.

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Methods	Description
IReplObjHandler::DeleteObj	Informs the ActiveSync service provider that an object should be deleted.
IReplObjHandler::GetPacket	ActiveSync service provider implements this method to deserialize an object into one or more packets. These packets are sent between the Windows CE-based device and the desktop computer by the ActiveSync service provider.
IReplObjHandler::Reset	Resets the ActiveSync service provider so all the resources

	that the ActiveSync service provider used during the serialization or deserialization are freed
IReplObjHandler::SetPacket	ActiveSync service provider implements this method to serialize one or more packets into an object. These packets are guaranteed to be in the same order as when they are sent.
IReplObjHandler::Setup	Sets up the ActiveSync service provider so it is ready to serialize or deserialize an object.
IUnknown::AddRef	Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a specified object.
IUnknown::QueryInterface	Returns a pointer to a specified interface on an object to which a client currently holds an interface pointer. This method must call IUnknown::AddRef on the pointer it returns.
IUnknown::Release	Decrements the reference count for the calling interface on an object. If the reference count on the object falls to 0, the object is freed from memory.

Remarks

The IReplObjHandler : IUnknown interface encapsulates all functions needed to serialize or deserialize the objects. Any object can be deserialized into one or more data packets of any size. An ActiveSync service provider determines the number of packets and their sizes. These packets are exchanged between the Windows CE-based device and the desktop computer. The receiver of these packets is guaranteed to receive them in the exact same order as they are sent and the receiver can then serialize these packets back into an object.

IReplObjHandler::DeleteObj

The IReplObjHandler::DeleteObj method informs the ActiveSync service provider that an object should be deleted.

Syntax
```
HRESULT DeleteObj(
    PREPLSETUP pSetup
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

Setup

Pointer to a REPLSETUP structure.

Return Values

NOERROR

The operation was successful.

Remarks

The IReplObjHandler::DeleteObj method is called whenever the ActiveSync service manager determines that an object needs to be deleted. Note that Setup and Reset are not called before and after this method. The ActiveSync service provider should delete the object specified in the given REPLSETUP structure.

See Also

IReplObjHandler

IReplObjHandler::GetPacket

The ActiveSync service provider implements IReplObjHandler::GetPacket to deserialize an object into one or more packets. These packets are sent between the Windows CE-based device and the desktop computer by the ActiveSync service provider.

Syntax
```
HRESULT GetPacket(
LPBYTE *lppbData,
DWORD *pcbData,
DWORD cbRecommend
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

lppbData

Pointer to a pointer of the outgoing packet.

pcbData
Pointer to a DWORD for the packet size.
cbRecommend
Recommended maximum size of the packet.

Return Values NOERROR
The operation successfully created one packet.
RERR_BAD_OBJECT
The operation failed to create one object. If the receiver does receive some of the earlier packets, they should be discarded.
RWRN_LAST_PACKET
A packet was successfully created, and it is the last one for the object.

Remarks During a deserialization of an object, the ActiveSync service manager calls the IreplObjHandler::GetPacket method continuously until RWRN_LAST_OBJECT or an error value is returned. The ActiveSync service provider determines how many packets are to be sent and the sizes of each packet. For efficiency, a packet size is recommended to be less than 8,000 bytes in size.

Allocation and deallocation of memory for the packet is the responsibility of the ActiveSync service provider. An ActiveSync service provider sets *lppbData* to that pointer and sets *pcbData* with the packet size. Typically, an ActiveSync service provider allocates a piece of memory of a known size in IReplObjHandler::Setup and frees it in IReplObjHandler::Reset.

See Also IReplObjHandler::SetPacket

IReplObjHandler::Reset

The IReplObjHandler::Reset method prompts the ActiveSync service provider to reset or free any resources used during the serialization or deserialization of an object.

Syntax
```
HRESULT Reset(
PREPLSETUP pSetup
);
```

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters *pSetup*
Pointer to a REPLSETUP structure.

Return Values NOERROR
The operation was successful.

Remarks The IReplObjHandler::Reset method is called once per object.

See Also IReplObjHandler::Setup

IReplObjHandler::SetPacket

The ActiveSync service provider implements SetPacket to serialize one or more packets into an object. These packets are guaranteed to be in the same order as when they are sent.

Syntax
```
HRESULT SetPacket(
LPBYTE lbpData,
DWORD cbData
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters
lpbData
Pointer to the incoming packet.
cbData
Stores the packet size.

Return Values NOERROR
The packet was successfully used to deserialize the object.
RERR_SKIP_ALL
Failed to apply the packet toward the object; skip all remaining packets for the object.

Remarks The IReplObjHandler::SetPacket method is called continuously until the last packet is received. These packets are guaranteed to be received in the same number and order as they are created by IReplObjHandler::GetPacket.

See Also IReplObjHandler::GetPacket

IReplObjHandler::Setup

The IReplObjHandler::Setup method sets up the ActiveSync service provider so it is ready to serialize or deserialize an object.

Syntax
HRESULT Setup (
PREPLSETUP *pSetup*
);

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

pSetup

Pointer to a REPLSETUP structure, which contains information about the object to be serialized or deserialized.

Remarks

The IReplObjHandler::Setup method is called once per object. Necessary data is stored in the passed REPLSETUP structure.

See Also

REPLSETUP

IReplStore : IUnknown

The IReplStore : IUnknown interface implements all required functions related to the store.

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

IReplStore Methods	Description
IReplStore::ActivateDialog	Activates an ActiveSync service provider-specific dialog box.
IReplStore::BytesToObject	Converts an array of bytes to a HREPLOBJ, which can be either a HREPLITEM or HREPLFLD, when loading.
IReplStore::CompareItem	Compares the specified handles using entry identifiers, such as file names or record numbers.
IReplStore::CompareStoreIDs	Compares two store identifiers to determine of they are equal.
IReplStore::CopyObject	Copies one HREPLOBJ, which can be either a HREPLITEM or HREPLFLD, over to another.

IReplStore::FindFirstItem	Returns a new HREPLITEM of the first object in the given folder, if there's any.
IReplStore:FindItemClose	Completes the Find operation in the given folder.
IReplStore::FindNextItem	Returns a new HREPLITEM of the next object in the given folder, if there's any.
IReplStore::FreeObject	Frees the specified HREPLOBJ handle.
IReplStore::GetConflictInfo	Gets information about two conflicting objects.
IReplStore::GetFolderInfo	Returns a HREPLFLD for folder, given the object type name. Also returns a pointer to the IReplObjHandler of the given object type.
IReplStore::GetObjTypeUIData	Sends user interface (UI)-related data about an object type to the ActiveSync service manager.
IReplStore:GetStoreInfo	Gets information about the current store instance.
IReplStore::Initialize	Initializes the ActiveSync service provider.
IReplStore::IsFolderChanged	Determines if any object in a specified folder has been changed since the method was last called.
IReplStore::IsItemChanged	Determines if the item has changed.
IReplStore::IsItemReplicated	Determines if the item should be replicated using ActiveSync service provider-defined rules.
IReplStore::IsValidObject	Determines if the specified handles are valid.
IReplStore::ObjectToBytes	Converts the HREPLOBJ, which can be either a HREPLITEM or HREPLFLD, to an array of bytes when saving.
IReplStore::RemoveDuplicates	Finds and removes duplicated objects from the store.
IReplStore::ReportStatus	ActiveSync service manager is reporting to the store about the status of the synchronization.

IReplStore::UpdateItem	Updates the object's time stamp, change number, and other information that is stored in the specified handle.
IUnknown::AddRef	Increments the reference count for an interface on an object. It should be called for every new copy of a pointer to an interface on a specified object.
IUnknown::QueryInterface	Returns a pointer to a specified interface on an object to which a client currently holds an interface pointer. This method must call IUnknown::AddRef on the pointer it returns.
IUnknown::Release	Decrements the reference count for the calling interface on an object. If the reference count on the object falls to 0, the object is freed from memory.

Remarks

The IReplStore : IUnknown interface encapsulates all functions needed to access the objects in the store. A handle of type HREPLITEM identifies each object in the store.

IReplStore::ActivateDialog

The IReplStore::ActivateDialog method activates an ActiveSync service provider-specific dialog box.

Syntax

```
HRESULT ActivateDialog(
UINT uDlg,
HWND hwndParent,
HREPLFLD hFolder,
IEnumReplItem * penum
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

uDlg

Identifies the dialog box to be activated.

hwndParent
Handle to the window that should be used as parent for the dialog box.
hFolder
Handle to a folder.
penum
Pointer to an enumerator of HREPLITEM for objects stored in the folder.

Return Values NOERROR
User selected OK to save the changes made.
RERR_CANCEL
User selected CANCEL to ignore the changes made.
RERR_SHUT_DOWN
User selected OK to save the changes made. The ActiveSync service manager must be closed now because of these changes.
RERR_UNLOAD
User selected OK to save the changes made. Replication modules must be unloaded so the change can take effect.
E_NOTIMPL
The requested dialog box is not implemented.

Remarks The IReplStore::ActivateDialog method is used to activate dialog boxes options for each object type. ReplDialogs contains the list of dialog boxes that can be activated. An ActiveSync service provider can return E_NOTIMPL if it does not implement a particular dialog box. An enumerator of the HREPLITEM contained in the specified folder is passed in. The ActiveSync service provider should use this enumerator to enumerate all items in the folder.

See Also IReplStore

IReplStore::BytesToObject

The IReplStore::BytesToObject method converts an array of bytes to an HREPLOBJ, which can be HREPLITEM or HREPLFLD, when loading.

Syntax
```
HREPLOBJ BytesToObject(
LPBYTE lpb,
UINT cb
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *lbp*

Pointer to a buffer where the array of bytes should be stored. This parameter can be NULL.

cb

Size of the buffer.

Remarks The IReplStore::BytesToObject method is used to convert a series of bytes into an item or folder handle. BytesToObject returns the new handle.

See Also IReplStore::ObjectToBytes

IReplStore::CompareItem

The IReplStore::CompareItem method compares the specified handles using entry identifiers, such as file names or record numbers.

Syntax

```
int CompareItem(
HREPLITEM hItem1,
HREPLITEM hItem2
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hItem1*

Handle to the first object. The ActiveSync service manager guarantees this handle is one of those returned by FindFirstItem or FindNextItem.

hItem2

Handle to the second object. The ActiveSync service manager guarantees this handle is one of those returned by FindFirstItem or FindNextItem.

Return Values 0

These two handles represent the same object.

1

The first object is bigger than the second object.

-1

The first object is smaller than the second object.

See Also HREPLITEM, IReplStore::IsItemChanged

IReplStore::CompareStoreIDs

The IReplStore::CompareStoreIDs method compares two store identifiers to determine if they are equal.

Syntax

```
HRESULT CompareStoreIDs(
LPBYTE lpbID1,
UINT cbID1,
LPBYTE lpbID2,
UINT cbID2
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

lbpID1
Pointer to the first store identifier.

cbID1
Size of the first store identifier.

lpbID2
Pointer to the second store identifier.

cbID2
Size of the second store identifier.

Return Values

0
These store identifiers represent the same store.

1
The first store is bigger than the second store.

-1
The first store is smaller than the second store.

Remarks

Replication calls the IReplStore::CompareStoreIDs method whenever it needs to know if the current store is different than the one it last replicated with. The store identifiers passed are always obtained from the STOREINFO structure set by the IReplStore::GetStoreInfo method.

See Also

IReplStore::GetStoreInfo, STOREINFO

IReplStore::CopyObject

The IReplStore::CopyObject method copies one HREPLOBJ, which can be either a HREPLITEM or HREPLFLD, over to another.

Syntax
```
BOOL CopyObject(
HREPLOBJ hObjSrc,
HREPLOBJ hObjDst
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hObjSrc
Handle to the source.

hObjDst
Handle to the destination.

Return Values

TRUE
The operation was successful.

FALSE
The operation failed. A possible reason is that the two handles are of different types or of different sizes.

Remarks

The IReplStore::CopyObject method is used to copy the contents of a specified handle to another. Any resource allocated in the source must be freed before they are overwritten, and any resource in the destination should be reset so it is not freed after the assignment to the source. CopyObject is always called when the ActiveSync service manager detects that an object has been modified since the last replication and its contents must therefore be updated from the modified handle returned by the ActiveSync service provider from FindNextItem or FindNextItem.

See Also

IReplStore

IReplStore::FindFirstItem

The IReplStore::FindFirstItem method returns a new handle to the first object in a specified folder, if there is any.

Syntax
```
HRESULT FindFirstItem(
HREPLFLD hFolder,
HREPLITEM *phItem,
BOOL *pfExist
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hFolder*
Handler to a folder.
phItem
Output pointer to a handle of the first object in the folder.
pfExist
Output pointer to a Boolean value that is set to TRUE if there is an object in the folder.

Return Values E_FAIL
There are problems with the enumeration. Replication should ignore the folder.
NOERROR
A new HREPLITEM was created for the first object in the folder and its pointer has been returned.

Remarks The IReplStore::FindFirstItem method works together with FindNextItem and FindItemClose to enumerate all items in a specified folder. FindFirstItem and FindNextItem are the only methods in IReplStore that can create HREPLITEM for the items. All HREPLITEM structures passed by the ActiveSync service manager are guaranteed to be originally created from these two methods. It is possible that, before FindItemClose is called, a different thread calls methods like DeleteObject that write to the store. Therefore, it is important for the ActiveSync service provider to have some sort of thread synchronization between this method and the methods that write to the store. A typical ActiveSync service provider would use critical section to make sure that, during the time between calls to FindFirstItem and FindItemClose, no write to the store is permitted.

See Also HREPLITEM, IReplStore::FindItemClose, IReplStore::FindNextItem

IReplStore::FindItemClose

The IReplStore::FindItemClose method completes the folder enumeration.

Syntax
```
HRESULT FindItemClose(
HREPLFLD hFolder
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hFolder*
Handle for the folder being enumerated.

Return Values NOERROR
The operation was successful.

Remarks The IReplStore::FindItemClose method works with FindFirstItem and FindNextItem to enumerate all items in a specified folder. An ActiveSync service provider can do whatever it needs to complete the enumeration, for example, free memory and delete temporary objects.

See Also HREPLITEM, IReplStore::FindFirstItem, IReplStore::FindNextItem

IReplStore::FindNextItem

The IReplStore::FindNextItem method returns a new item handle to the next object in a specified folder, if there is any.

Syntax
```
HRESULT FindNextItem(
HREPLFLDF hFolder,
HREPLITEM *phItem
BOOL *pfExist
);
```

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters *hFolder*
Handle to a folder.
phItem
Output pointer to a handle of the next object in the folder.
pfExist
Output pointer to a Boolean value that is set to TRUE if there is an object in the folder.

Return Values E_FAIL
There are problems with the enumeration. Replication should ignore the folder.
NOERROR
A new HREPLITEM was created for the next object in the folder and its pointer has been returned.

Remarks The IReplStore::FindNextItem method works with FindFirstItem and FindItemClose to enumerate all items in a specified folder. FindNextItem and FindFirstItem are the only methods in

IReplStore that can create HREPLITEM structures for the objects. All HREPLITEM structures passed by the ActiveSync service manager are guaranteed to be originally created from these two methods.

See Also HREPLITEM, IReplStore::FindFirstItem, IReplStore::FindItemClose

IReplStore::FreeObject

The IReplStore::FreeObject method frees the specified HREPLOBJ handle.

Syntax
```
void FreeObject(
HREPLOBJ hObject
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hObject

Pointer to the handle of an object whose contents need to be freed.

Return Values None.

Remarks The IReplStore::FreeObject method is used to free any memory pointers or delete any temporary objects that might have been created during the life of the handle and must be freed when the handle dies. This handle could either be an HREPLITEM or HREPLFLD structure.

See Also IReplStore

IReplStore::GetConflictInfo

The IReplStore::GetConflictInfo method gets information about two conflicting objects.

Syntax
```
HRESULT GetConflictInfo(
PCONFINFO pConfInfo
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters *pConfInfo*
Pointer to the CONFINFO structure.

Return Values NOERROR
Information was retrieved successfully.
RERR_IGNORE
This conflict should be ignored. The objects are identical.

See Also IReplStore

IReplStore::GetFolderInfo

The IReplStore::GetFolderInfo method creates a new HREPLFLD of a folder for the specified object type name and returns a pointer to the IReplObjHandler interface that is used to serialize and deserialize all items in this folder.

Syntax
```
HRESULT GetFolderInfo(
LPSTR lpszName,
HREPLFLD *phFolder,
IUnknown **ppObjHandler
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters *lpszName*
Name of the object type as taken from the registry.
phFolder
Output pointer to the handle of the folder.
ppObjHandler
Output pointer to a pointer to the IReplObjHandler interface.

Return Values NOERROR
The operation was successful.

Remarks The IReplStore::GetFolderInfo method is the only method in IReplStore that creates or modifies a HREPLFLD structure for the folder. The ActiveSync service manager calls this method to get a folder handle for the specified object type. Object types are configured into the registry, where object type name and other relevant information about an object type are stored. Note that

the handle pointed to by *phFolder* may or may not be NULL when called. If *phFolder* points to a handle that has a NULL value, the ActiveSync service provider should create a new handle for the specified folder. If *phFolder* points to a pointer that has a value, the ActiveSync service provider should modify the data indicated by this handle.

See Also IReplStore

IReplStore::GetObjTypeUIData

The IReplStore::GetObjTypeUIData method sends user interface (UI)-related data about an object type to the ActiveSync service manager.

Syntax
```
HRESULT GetObjTypeUIData(
HREPLFLD hFolder,
POBJUIDATA pData
);
```

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters

hFolder
Input parameter. Pointer to a handle of a folder that contains the items.

pData
Output parameter. Pointer to an OBJUIDATA structure.

Return Values

NOERROR
User selected OK to save the changes made.

E_OUTOFMEMORY
The operation was unable to load required UI resources.

See Also IReplStore

IReplStore::GetStoreInfo

The IReplStore::GetStoreInfo method gets information about the current store instance.

Syntax
```
HRESULT GetStoreInfo(
PSTOREINFO pInfo
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters *pInfo*
Pointer to the STOREINFO structure.

Return Values NOERROR
The STOREINFO structure was successfully returned.
E_INVALIDARG
The value of *cbStruct* is not expected.
E_POINTER
The store is not initialized or there is a problem getting the required store identifier or *lpbStored* is NULL.
E_OUTOFMEMORY
The value of *cbMaxStoreId* is too small. The size of the identifier is set in *cbStoreId* upon return.

Remarks The ActiveSync service manager calls the IReplStore::GetStoreInfo method with *lpbStoreId* set to NULL for the first time. The ActiveSync service provider should then set *cbStoreId* to the size of the store identifier. Replication then calls GetStoreInfo again with an allocated buffer and the size stored in *cbMaxStoreId*.

See Also STOREINFO

IReplStore::Initialize

The IReplStore::Initialize method initializes the IReplStore ActiveSync service provider.

Syntax
```
HRESULT Initialize(
    IReplNotify *pReplNotify
    UINT uFlags
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters *pReplStatus*
Pointer to the IReplNotify interface. This parameter must be 0.
uFlags
Flags passed to the store by the ActiveSync service manager. Possible values include the following:

ISF_SELECTED_DEVICE
Set if the store is initialized for the selected device; otherwise, it is initialized for the connected device.
ISF_REMOTE_CONNECTED
Set if the store is initialized during the remote connection; all user interface (UI) should be suppressed.

Return Values NOERROR
The operation was successful.

See Also IReplStore

IReplStore::IsFolderChanged

The IReplStore::IsFolderChanged method determines if any object in a specified folder has been changed since the method was last called.

Syntax
```
HRESULT IsFolderChanged(
HREPLFLD hFolder,
BOOL *pfChanged
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters
hFolder
Handle to a folder.
pfChanged
Pointer to a Boolean value that is set to TRUE if folder is changed.

Return Values NOERROR
The operation completed successfully. The *pfChanged* parameter is set to TRUE if the folder is changed, or FALSE otherwise.
RERR_SHUT_DOWN
There was a serious error, and the ActiveSync service provider should shut down immediately.
RERR_UNLOAD
There was a less serious error, and replication modules must be unloaded.
RERR_STORE_REPLACED
The complete store was replaced.

Remarks — If the ActiveSync service provider wants real-time synchronization to be simulated; see GetStoreInfo. The ActiveSync service manager calls the IReplStore::IsFolderChanged method once the timer is up to see if it needs to scan the store further to pick up any changes. This is used to reduce the number of scans replication has to make to the store. An ActiveSync service provider should return TRUE if it does not need to implement this method.

See Also — IReplStore::GetStoreInfo, STOREINFO

IReplStore::IsItemChanged

The IReplStore::IsItemChanged method determines if the object has changed.

Syntax

```
BOOL IsItemChanged(
HREPLFLD hFolder,
HREPLITEM hItem,
HREPLITEM hItemComp
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hFolder
Handle to the folder or container that stores the object.

hItem
Handle to the object.

hItemComp
Handle to the object used for comparison.

Return Values

FALSE
The object has not been changed.

TRUE
The object has changed.

Remarks — If *hItemComp* is not NULL, the ActiveSync service provider should check the data (time stamp, change number) in *hItem* with *hItemComp*. If *hItemComp* is NULL, the ActiveSync service provider should get the data by opening the object and comparing it with the data in *hItem*.

See Also — HREPLITEM, IReplStore::CompareItem

IReplStore::IsItemReplicated

The IReplStore::IsItemReplicated method determines if an item should be replicated using ActiveSync service provider-defined rules.

Syntax

```
BOOL IsItemReplicated(
HREPLIFLD hFolder,
HREPLITEM hItem
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hFolder

Handle to the folder or container that stores the object.

hItem

Handle to the object. This parameter can be NULL, in which case, IsItemReplicated should determine if the specified folder should be replicated.

Return Values

FALSE

The object should not be replicated.

TRUE

The object should be replicated.

Remarks

If the ActiveSync service provider requires that some objects on the desktop computer should not be replicated, it can use the IReplStore::IsItemReplicated method to tell the ActiveSync service manager to ignore these objects. The ActiveSync service provider can design its own rules and store it using the handle of the folder. If all objects should be replicated, the ActiveSync service provider can return TRUE in all calls.

See Also

IReplStore

IReplStore::ObjectToBytes

The IReplStore::ObjectToBytes method converts the HREPLOBJ, which can be either a HREPLITEM or HREPLFLD, to an array of bytes when saving.

Syntax

```
UINT ObjectToBytes(
HREPLOBJ hObject,
LPBYTE lpb
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
hObject
Handle to an object.
lpb
Handle to a buffer where the array of bytes should be stored. This parameter can be NULL.

Return Values Number of bytes in the array.

Remarks The IReplStore::ObjectToBytes method is used to save the data represented by a handle to disk. The ActiveSync service manager calls ObjectToBytes first with *lpb* set to NULL. The ActiveSync service provider should then return the size required, followed by the ActiveSync service manager calling ObjectToBytes with a *lpb* parameter pointing to a buffer large enough for the array.

See Also IReplStore::BytesToObject

IReplStore::IsValidObject

The IReplStore::IsValidObject method determines if the specified handles are valid.

Syntax
```
HRESULT IsValidObject(
HREPLFLD hFolder,
HREPLITEM hItem,
UINT, uFlags
);
```

At a Glance
Header file: Cesync.h
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
hFolder
Handle to a folder. This parameter can be NULL.
hItem
Handle to an item. This parameter can be NULL.
uFlags
Reserved. Must be 0.

Return Values NOERROR
The specified handles are all valid.
RERR_CORRUPT
The data in the specified handle is corrupted.

RERR_OBJECT_DELETED
The object identified by the handle is no longer in the store.

Remarks — The IReplStore::IsValidObject method is used to determine if the specified handles are valid. The ActiveSync service provider should check both *hFolder* and *hItem* to determine if either of them is not NULL.

See Also — IReplStore

IReplStore::RemoveDuplicates

The IReplStore::RemoveDuplicates method finds and removes duplicated objects from the store.

Syntax

```
HRESULT Remove Duplicates(
LPSTR lpszObjType,
UINT uFlags
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

lpszObjType
Pointer to the name of the object type for which this operation is intended. This parameter is NULL if all object types should be checked.

uFlags
Reserved. Always 0.

Return Values

NOERROR
The operation completed successfully and there is no need to restart replication to pick up the deletions.

RERR_RESTART
The operation completed successfully and replication should be restarted to pick up the deletions.

E_NOTIMPL
The ActiveSync service provider does not support this operation.

Remarks — Occasionally, the ActiveSync service manager might need to prompt an ActiveSync service provider to scan all objects in the store to check for duplicates and give the user a chance to remove them. The ActiveSync service provider should return E_NOTIMPL if it chooses not to implement this functionality. Otherwise, the ActiveSync service provider should perform the check and remove and return NOERROR or RERR_RESTART if successful. In this case, replication does not call the IReplStore::RemoveDuplicates method again until necessary. It should return all other error values if, for some reason, operation cannot be performed at that time. In this case, replication calls RemoveDuplicates again at the end of the next synchronization.

See Also — IReplStore

IReplStore::ReportStatus

ActiveSync service manager calls the IReplStore::ReportStatus method to get information on the synchronization status.

Syntax

```
HRESULT ReportStatus(
HREPLFLD hFolder,
HREPLITEM hItem,
UINT uStatus,
UINT uParam
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hFolder
Handle to the folder this status applies to. This parameter is NULL if status applies to all folders.

hItem
Handle to the object this status applies to. This parameter is NULL if status applies to all objects.

uStatus
Status code. Possible values include the following:

RSC_BEGIN_SYNC
Synchronization is about to start; *uReserved* is a combination of the following bit flags:

BSF_AUTO_SYNC
Synchronization is started as a result of changes while "autosync on change" is turned on.

BSF_REMOTE_SYNC
Consistent with RSC_REMOTE_SYNC, set if synchronization is done remotely.

RSC_END_SYNC
Synchronization has ended.

RSC_BEGIN_CHECK
The ActiveSync service manager is about to call FindFirstItem and FindNextItem.

RSC_END_CHECK
The ActiveSync service manager has completed all enumeration calls and FindItemClose has been called.

RSC_DATE_CHANGED
The user has changed the system date. This code is called on every existing object in the store to give the ActiveSync service provider a chance to reset the date-dependent synchronization options. For example, if an ActiveSync service provider wants to synchronize files that are modified in the last two weeks, it can respond to this code to reset the enable bit for each item. When IsItemReplicated is called later, it re-evaluates the items based on the new date.

RSC_RELEASE
The ActiveSync service manager is about to release the IReplStore object. This is called before the final IReplStore::Release call.

RSC_REMOTE_SYNC
If *uParam* is TRUE, the ActiveSync service manager is about to start remote synchronization. The ActiveSync service provider should not show any UI that requires user interaction from now on until this status code is used again with *uParam* equal to FALSE.

RSC_INTERRUPT
ActiveSync service manager is about to interrupt the current operation.

The following values of *uParam* are defined only for RSC_INTERRUPT:

PSA_RESET_INTERRUPT
This flag is set if the interrupt state is being cleared; that is, normal operation is resuming.

PSA_SYS_SHUTDOWN
User has shut down the Windows operating system.

RSC_BEGIN_SYNC_OBJ
Synchronization is about to start on an object type. *uReserved* is a combination of bit flags; see RSC_BEGIN_SYNC.

RSC_END_SYNC_OBJ
Synchronization is about to end on an object type.
RSC_OBJ_TYPE_ENABLED
Synchronization of the specified object is enabled; *hFolder* is a pointer to a string (object type name).
RSC_OBJ_TYPE_DISABLED
Synchronization of the specified object is disabled; *hFolder* is a pointer to a string (object type name).
RSC_BEGIN_BATCH_WRITE
A series of SetPackets is called on a number of objects. This is the time for ActiveSync service provider to start a transaction.
RSC_END_BATCH_WRITE
RSC_BEGIN_BATCH_WRITE has ended. This is the time for the ActiveSync service provider to commit the transaction.
RSC_CONNECTION_CHG
The connection status has changed. *uParam* is TRUE if a connection has been established; otherwise, it is FALSE.
RSC_WRITE_OBJ_FAILED
There was a failure while writing to an object on the device. *uParam* is the HRESULT code.
RSC_DELETE_OBJ_FAILED
There was a failure while deleting an object on the device. *uParam* is the HRESULT code.

uParam
Additional information about the status, based on *uStatus* code.

Return Values NOERROR
The process indicated by *uStatus* is successful.
E_FAIL
The process indicated by *uStatus* has failed or encountered problems.

Remarks The Active Sync service provider can return NOERROR for all cases if it is not interested.

This is an application programming interface (API) exported by the Store.dll for the synchronization engine.

See Also IReplStore

IReplStore::UpdateItem

The IReplStore::UpdateItem method updates the object's time stamp, change number, and other information that is stored in the specified handle.

Syntax

```
void UpdateItem(
HREPLFLD hFolder,
HREPLITEM hItemDst,
HREPLITEM hItemSrc
);
```

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters

hFolder
Handle to a folder that stores the item.

hItemDst
Handle to the destination item.

hItemSrc
Handle to the source item; could be NULL.

Return Values None.

Remarks

The ActiveSync service manager calls the IReplStore::UpdateItem method to update the relevant information, such as time stamp or change number, in the specified handle. If a source handle is specified, the ActiveSync service provider should copy the information over; otherwise, the ActiveSync service provider should open the object, then get the object's information and store it in the destination handle.

See Also IReplStore

IEnumReplItem

The IEnumReplItem interface enables enumeration of a collection of items.

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Method	Description
IEnumReplItem::Clone	Creates a copy of the current state of enumeration.
IEnumReplItem::GetFolder Handle	Gets a handle to the folder (HREPLFD) that is currently being enumerated.
IEnumReplItem::Next	Attempts to advance to the next item in the enumeration sequence.
IEnumReplItem::Reset	Resets the enumeration sequence to the beginning.
IEnumReplItem::Skip	Attempts to skip over the next item in the enumeration sequence.

IEnumReplItem::Clone

The IEnumReplItem::Clone method creates a copy of the current state of enumeration.

Syntax

```
HRESULT Clone(
IEnumReplItem FAR * FAR * ppEnum,
);
```

At a Glance

Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

ppEnum

Pointer to the place to return the cloned enumerator. The type of *ppEnum* is the same as the enumerator name. For example, if the enumerator name is IEnumFORMTETC, *ppEnum* is of type IEnumFORMATETC.

Return Values

E_OUTOFMEMORY

Out of memory.

E_INVALIDARG

Value of *ppEnum* is invalid.

E_UNEXPECTED

An unexpected error occurred.

IEnumReplItem::GetFolderHandle

The IEnumReplItem::GetFolderHandle method gets a handle to the folder (HREPLFLD) that is currently being enumerated.

Syntax hHREPLFLD GetFolderHandle ();

At a Glance
Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Return Values Returns the handle to the folder (HREPLFLD) that is being enumerated.

IEnumReplItem::Next

The IEnumReplItem::Next method attempts to advance to the next item in the enumeration sequence.

Syntax
```
HRESULT Next(
unsigned long celt,
HREPLITEM *phItem,
unsigned long FAR *pCeltFetched,
);
```

At a Glance
Header file:	Cesync.h
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

celt
Specifies the number of elements to return. If the number of elements requested is more than remains in the sequence, only the remaining elements are returned. The number of elements returned is passed through the *pCeltFetched* parameter, unless it is NULL.

phItem
Pointer to the structure in which to return the elements.

pCeltFetched
Pointer to the number of elements actually returned in **phItem*. The *pCeltFetched* parameter cannot be NULL if *celt* is greater than one. Likewise, if *pCeltFetched* is NULL, *celt* must be one.

Return Values

S_OK
Returned the requested number of elements; *phItem* is set if non-NULL. All requested entries are valid.

S_FALSE
Returned fewer elements than requested in *celt*. In this case, unused slots in the enumeration are not set to NULL and **phItem* holds the number of valid entries, even if zero is returned.

E_OUTOFMEMORY
Out of memory.

E_INVALIDARG
The value of *celt* is invalid.
E_UNEXPECTED
An unexpected error occurred.

IEnumReplItem::Reset

The IEnumReplItem::Reset method resets the enumeration sequence to the beginning.

Syntax HRESULT Reset():

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Return Values S_OK
The enumeration sequence was reset to the beginning.
S_FALSE
The enumeration sequence was not reset to the beginning.

IEnumReplItem::Skip

The IEnumReplItem::Skip method attempts to skip over the next item in the enumeration sequence.

Syntax
```
HRESULT Skip(
unsigned long celt,
);
```

At a Glance		
	Header file:	Cesync.h
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters *celt*
Specifies the number of elements to be skipped.

Return Values S_OK
The number of elements skipped is equal to *celt*.
S_FALSE
The number of elements skipped is fewer than *celt*.
S_OUTOFMEMORY
Out of memory.
E_INVALIDARG
The value of *celt* is invalid.
E_UNEXPECTED
An unexpected error occurred.

Detailed Description of a Database API

Chapter 19

Fsdbase Component: Functions

CeCreateDatabase

The CeCreateDatabase function creates a new database. A RAPI version of this function exists and is also called CeCreateDatabase.

Syntax

CEOID CeCreateDatabase(LPWSTR *lpszName*, DWORD *dwDbaseType*, WORD *wNumSortOrder*, SORTORDERSPEC **rgSortSpecs*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

lpszName

Pointer to a null-terminated string that specifies the name for the new database. The name can have up to 32 characters, including the terminating null character. If the name is too long, it is truncated.

dwDbaseType

Type identifier for the database. This is an application-defined value that can be used for any application-defined purpose. For example, an application can use the type identifier to distinguish address book data from to-do list data or use the identifier during a database enumeration sequence. See CeFindFirstDatabase for details. The type identifier is not meant to be a unique identifier for the database. The system does not use this value.

wNumSortOrder

Number of sort orders active in the database, with four being the maximum number. This parameter can be zero if no sort orders are active.

rgSortSpecs

Pointer to an array of actual sort order descriptions. The size of the array is specified by *wNumSortOrder*. This parameter can be NULL if *wNumSortOrder* is zero.

Remarks

Because sort orders increase the system resources needed to perform each insert and delete operation, keep the number of sort orders to a minimum. However, try not to specify too few sort orders. If you do, you can use the CeSetDatabaseInfo function to change the sort order later; however, this function is even more expensive in terms of system resources.

Return Values If the function succeeds, the return value is the object identifier of the newly created database – not a handle to an open database. If the function fails, the return value is NULL. To get extended error information when within a CE program, call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_DISK_FULL
The object store does not contain enough space to create the new database.

ERROR_INVALID_PARAMETER
A parameter was invalid.

ERROR_DUP_NAME
A database already exists with the specified name.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegCreateDatabase function.

See Also CeDeleteDatabase, CeOidGetInfo, CeOpenDatabase, CeSetDatabaseInfo, SORTORDERSPEC

CeDeleteDatabase

The CeDeleteDatabase function removes a database from the object store. A RAPI version of this function exists and is also called CeDeleteDatabase.

Syntax BOOL CeDeleteDatabase(CEOID *oidDbase*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters *oidDbase*
Object identifier of the database to be deleted.

Return Values If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. To get extended error information when within a CE program call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_INVALID_PARAMETER
A parameter was invalid.

ERROR_SHARING_VIOLATION
Another thread has an open handle to the database.

Remarks The CeDeleteDatabase function deletes a database, including all records in the database.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegDeleteDatabase function.

See Also CeCreateDatabase, CeOidGetInfo

CeDeleteRecord

The CeDeleteRecord function deletes a record from a database. A RAPI version of this function exists and is also called CeDeleteRecord.

Syntax BOOL CeDeleteRecord(HANDLE *hDatabase*, CEOID *oidRecord*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters *hDatabase*
Handle to the database from which the record is to be deleted. The database must be open. Open a database by calling the CeOpenDatabase function.
oidRecord
Object identifier of the record to be deleted; this is obtained from CeOpenDatabase.

Return Values If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. To get extended error information when within a CE program cell GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return ERROR_INVALID_PARAMETER if the handle or object identifier is invalid.

Remarks If the CEDB_AUTOINCREMENT flag was not specified when the database was opened, and the record being deleted is the current record, the next read operation that uses the database handle will fail. If the CEDB_AUTOINCREMENT flag was specified, the system automatically moves the current seek pointer forward by one.

When writing applications for Windows CE version 1.0, use the PegDeleteRecord function.

See Also CeOpenDatabase

CeFindFirstDatabase

The CeFindFirstDatabase function opens an enumeration context for all databases in the system. A RAPI version of this function exists and is also called CeFindFirstDatabase.

Syntax
At a Glance

HANDLE CeFindFirstDatabase(DWORD *dwDbaseType*);

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters *dwDbaseType*
Type identifier of the databases to enumerate. If this parameter is zero, all databases are enumerated.

Return Values If the function succeeds, the return value is a handle to an enumeration context. To find the next database of the given type, specify the handle in a call to the CeFindNextDatabase function. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information when within a CE program call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return ERROR_OUTOFMEMORY if no memory is available to allocate a database handle.

Remarks Use the CeCloseHandle function to close the handle returned by the CeFindFirstDatabase function.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegFindFirstDatabase function.

See Also CeFindNextDatabase, CeCloseHandle

CeFindNextDatabase

The CeFindNextDatabase function retrieves the next database in an enumeration context. A RAPI version of this function exists and is also called CeFindNextDatabase.

Syntax CEOID CeFindNextDatabase(HANDLE *hEnum*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters *hEnum*

Handle to an enumeration context; this handle is returned from CeFindFirstDatabase.

Return Values If the function succeeds, the return value is the object identifier of the next database to be enumerated. If no more databases are left to enumerate, or if an error occurs, the return value is zero. To get extended error information when within a CE program, call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_NO_MORE_ITEMS
The object store contains no more databases to enumerate.

ERROR_INVALID_PARAMETER
The *hEnum* parameter specified an invalid handle.

Remarks When writing applications for Windows CE version 1.0, use the PegFindNextDatabase function.

See Also CeFindFirstDatabase

CeOpenDatabase

The CeOpenDatabase function opens an existing database. A RAPI version of this function exists and is also called CeOpenDatabase.

Syntax HANDLE CeOpenDatabase(PCEOID *poid*, LPWSTR *lpszName*, CEPROPID *propid*, DWORD *dwFlags*, HWND *hwndNotify*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

poid

Pointer to the object identifier of the database to be opened. To open a database by name, set the value pointed to by *poid* to zero to receive the object identifier of the newly opened database when a database name is specified for *lpszName*.

lpszName

Pointer to the name of the database to be opened. This parameter is ignored if the value pointed to by *poid* is non-zero.

propid

Property identifier of the primary key for the sort order in which the database is to be traversed. All subsequent calls to CeSeekDatabase assume this sort order. This parameter can be zero if the sort order is not important.

dwFlags

Action flag. The following values are supported:

CEDB_AUTOINCREMENT

Causes the current seek position to be automatically incremented with each call to the CeReadRecordProps function.

0 (ZERO)

Current seek position is not incremented with each call to CeReadRecordProps.

hwndNotify

Handle to the window to which notification messages (DB_CEOID_*) will be posted if another thread modifies the given database while you have it open. This parameter can be NULL if you do not need to receive notifications.

Return Values

If the function succeeds, the return value is a handle to the open database. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information when within a CE program cell GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_INVALID_PARAMETER

A parameter was invalid.

ERROR_FILE_NOT_FOUND
No database exists with the specified name. This value applies only if the value pointed to by *poid* was set to NULL when the function was called.

ERROR_NOT_ENOUGH_MEMORY
No memory was available to allocate a database handle.

Remarks

Use the CeCloseHandle function to close the handle returned by the CeOpenDatabase function.

Unlike many other traditional databases, opening and closing a database does not imply any transactioning. In other words, the database is not committed at the closing – it is committed after each individual call.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegOpenDatabase function.

See Also

CeCloseHandle, CeCreateDatabase, CeSeekDatabase

CeReadRecordProps

The CeReadRecordProps function reads properties from the current record. A RAPI version of this function exists and is also called CeReadRecordProps.

Syntax

CEOID CeReadRecordProps(HANDLE *hDbase*, DWORD *dwFlags*, LPWORD *lpcPropID*, CEPROPID **rgPropID*, LPBYTE * *lplpBuffer*, LPDWORD *lpcbBuffer*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

hDbase
Handle to an open database. The database must have been opened by a previous call to the CeOpenDatabase function.

dwFlags
Read flags. The following value is supported:

CEDB_ALLOWREALLOC
The LocalAlloc function was used to allocate the buffer specified by the *lplpBuffer* parameter, and the server can reallocate the buffer if it is not large enough to hold the requested properties.

lpcPropID
Number of property identifiers in the array specified by the *rgPropID* parameter. If *rgPropID* is NULL, this parameter receives the number of properties retrieved.

rgPropID
Pointer to an array of property identifiers for the properties to be retrieved. If this parameter is NULL, CeReadRecordProps retrieves all properties in the record.

lplpBuffer
Address of a pointer to a buffer that receives the requested properties. If the *dwFlags* parameter includes the CEDB_ALLOWREALLOC flag, the buffer may be reallocated if necessary. If the CEDB_ALLOWREALLOC flag is specified and this parameter is NULL, the server uses the LocalAlloc function to allocate a buffer of the appropriate size in the caller's address space and returns a pointer to the buffer. Note that if the CEDB_ALLOWREALLOC flag is specified, it is possible for the value of this pointer to change even on failure. For example, the old memory might be freed and the allocation might then fail, leaving the pointer set to NULL.

lpcbBuffer
Pointer to a variable that contains the size, in bytes, of the buffer specified by the *lplpBuffer* parameter. When CeReadRecordProps returns, *lpcbBuffer* receives a value that indicates the actual size of the data copied to the buffer. If the buffer was too small to contain the data, this parameter can be used to calculate the amount of memory to allocate for the buffer if CEDB_ALLOWREALLOC was not specified.

Return Values If the function succeeds, the return value is the object identifier of the record from which the function read. If the functional fails, the return value is zero. To get extended error information when within a CE program, call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_INVALID_PARAMETER
A parameter was invalid.

ERROR_NO_DATA
None of the requested properties was found. The output buffer and the size are valid.

ERROR_INSUFFICIENT_BUFFER
The given buffer was not large enough, and the reallocation failed — if the CEDB_ALLOWREALLOC flag was specified. The *lpcbBuffer* parameter contains the required buffer size.

ERROR_KEY_DELETED
The record that was about to be read was deleted by another thread. If the current record was reached as a result of an autoseek, this error is not returned, and the next record is returned.

ERROR_NO_MORE_ITEMS
The current seek pointer is at the end of the database.

Remarks

The CeReadRecordProps function reads the specified set of properties from the current record. If the database was opened with the autoseek flag — that is, if the *dwFlags* parameter of CeOpenDatabase was set to CEDB_AUTOINCREMENT — CeReadRecordProps increments the seek pointer by one so that the next call reads the next record in the current sort order. That is, if the database was opened with a sort order active, then CeReadRecordProps will return the records in sorted order. If the database was not opened with a sort order active, then the order in which records are returned is not predictable.

Read all needed properties from the record in a single call. The entire record is stored in a compressed format, and each time a property is read it must be decompressed. All the properties are returned in a single marshaled structure, which consists of an array of CEPROPVAL structures, one for each property requested — or one for each property found if the application set the *rgPropID* parameter to NULL when calling the function.

If a property was requested, such as strings or blobs that are packed in at the end of the array, the pointers in the CEPROPVAL structures point into this marshaled structure. This means that the only memory that must be freed is the original pointer to the buffer passed in to the call. Even if the function fails, it may have allocated memory on the caller's behalf. Free the pointer returned by this function if the pointer is not NULL.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegReadRecordProps function.

See Also LocalAlloc, LocalFree, CeOpenDatabase, CeSeekDatabase, CEPROPVAL

CeSeekDatabase

The CeSeekDatabase function seeks the specified record in an open database. A RAPI version of this function exists and is also called CeSeekDatabase.

Syntax CEOID CeSeekDatabase(HANDLE *hDatabase*, DWORD *dwSeekType*, DWORD *dwValue*, LPDWORD *lpdwIndex*);

At a Glance

Header file:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

hDatabase
Handle to the open database in which to seek.

dwSeekType
Type of seek operation to perform. This parameter can be one of the following values:

CEDB_SEEK_CEOID
Seek until finding an object that has the given object identifier. The *dwValue* parameter specifies the object identifier. This type of seek operation is very efficient.

CEDB_SEEK_VALUESMALLER
Seek until finding the largest value that is smaller than the given value. If none of the records has a smaller value, the seek pointer is left at the end of the database and the function returns zero. The *dwValue* parameter is a pointer to a CEPROPVAL structure.

CEDB_SEEK_VALUEFIRSTEQUAL
Seek until finding the first value that is equal to the given value. If the seek operation fails, the seek pointer is left pointing at the end of the database, and the function returns zero. The *dwValue* parameter is a pointer to a CEPROPVAL structure.

CEDB_SEEK_VALUENEXTEQUAL
Starting from the current seek position, seek exactly one position forward in the sorted order and check if the next record is equal in value to the given value. If so, return the object identifier of this next record; otherwise, return zero and leave the seek pointer at the end of the database. This operation can be used in conjunction with the CEDB_SEEK_VALUEFIRSTEQUAL operation to enumerate all records with an equal value. The *dwValue* parameter specifies the value for which to seek.

CEDB_SEEK_VALUEGREATER
Seek until finding a value greater than or equal to the given value. If all records are smaller, the seek pointer is left at the end of the database and the function returns zero. The *dwValue* parameter is a pointer to a CEPROPVAL structure.

CEDB_SEEK_BEGINNING
Seek until finding the record at the given position from the beginning of the database. The *dwValue* parameter specifies the number of records to seek.

CEDB_SEEK_CURRENT
Seek backward or forward from the current position of the seek pointer for the given number of records. The *dwValue* parameter specifies the number of records from the current position. The function seeks forward if *dwValue* is a positive value, or backward if it is negative. A forward seek operation is efficient.

CEDB_SEEK_END
Seek backward for the given number of records from the end of the database. The *dwValue* parameter specifies the number of records.

dwValue
Value to use for the seek operation. The meaning of this parameter depends on the value of *dwSeekType*.

lpdwIndex
Pointer to a variable that receives the index from the start of the database to the beginning of the record that was found.

Return Values If the function succeeds, the return value is the object identifier of the record on which the seek ends. If the function fails, the return value is zero. To get extended error information when within a CE program call GetLastError. If within a RAPI program, call CeGetLastError, GetLastError and CeGetLastError may return ERROR_INVALID_PARAMETER if a parameter is invalid.

Remarks The CeSeekDatabase function always uses the current sort order as specified in the call to the CeOpenDatabase function. If the CEDB_AUTOINCREMENT flag was specified, an automatic seek of one from the current position is done with each read operation that occurs on the database.

Note that a seek can only be performed on a sorted property value. After creating a database (using CeCreateDatabase) and opening the database (using CeOpenDatabase), subsequent calls to CeSeekDatabase assume the sort order that was specified in the *propid* parameter of the call to CeOpenDatabase. Although property identifiers can be modified using CeWriteRecordProps, it is best to use the same property identifier for CeOpenDatabase that was used for the propid member of the SORTORDERSPEC structure that was passed in the call to CeCreateDatabase.

To enter negative values for the CEDB_SEEK_CURRENT case, cast a signed long. This changes the effective range on the record indexes to 31 bits from 32.

Multiple sort orders cannot be specified for a single property.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegSeekDatabase function.

See Also CeCreateDatabase, CeOpenDatabase, CEPROPVAL

CeSetDatabaseInfo

The CeSetDatabaseInfo function sets various database parameters, including the name, type, and sort-order descriptions. A RAPI version of this function exists and is also called CeSetDatabaseInfo.

Syntax BOOL CeSetDatabaseInfo(CEOID *oidDbase*, CEDBASEINFO **pNewInfo*);

At a Glance

Header File:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

oidDbase

Object identifier of the database for which parameters are to be set.

pNewInfo

Pointer to a CEDBASEINFO structure that contains new parameter information for the database. The wNumRecords member of the structure is not used.

Return Values

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. To get extended error information when within a CE program call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values:

ERROR_INVALID_PARAMETER

A parameter was invalid.

ERROR_DISK_FULL

The object store is full and any size changes required could not be accommodated. Changing sort orders can change the size of the stored records, though not by much.

ERROR_SHARING_VIOLATION

CeSetDatabaseInfo tried to remove a sort order that is being used by a currently open database.

Remarks

The CeSetDatabaseInfo function can be used to change the database parameters passed in while creating the database. Note that changing the sort order of the database can take several minutes. Before calling CeSetDatabaseInfo, an application should warn the user that this operation can be lengthy.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegSetDatabaseInfo function.

See Also

CeCreateDatabase, CEDBASEINFO, CeOidGetInfo

CeWriteRecordProps

The CeWriteRecordProps function writes a set of properties to a single record, creating the record if necessary. A RAPI version of this function exists and is also called CeWriteRecordProps.

Syntax CEOID CeWriteRecordProps(HANDLE *hDbase*, CEOID *oidRecord*, WORD *cPropID*, CEPROPVAL **rgPropVal*);

At a Glance

Header File:	Winbase.h
Component:	fsdbase
Platforms:	H/PC
Windows CE versions:	1.01 and later

Parameters

hDbase
Handle to an open database. The database must have been opened by a previous call to the CeOpenDatabase function.

oidRecord
Object identifier of the record to which the given properties are to be written. If this parameter is zero, a new record is created and filled in with the given properties.

cPropID
Number of properties in the array specified by the *rgPropVal* parameter. The *cPropID* parameter must not be zero.

rgPropVal
Pointer to an array of CEPROPVAL structures that specify the property values to be written to the given record.

Return Values If the function succeeds, the return value is the object identifier of the record to which the properties were written. If the function fails, the return value is zero. To get extended error information when within a CE program call GetLastError. If within a RAPI program, call CeGetLastError. GetLastError and CeGetLastError may return one of the following values::

ERROR_DISK_FULL
There was not enough space in the object store to write the properties.

ERROR_INVALID_PARAMETER
A parameter was invalid.

Remarks The CeWriteRecordProps function writes all the requested properties into the specified record. CeWriteRecordProps does not move the seek pointer.

To delete a property, set the CEDB_PROPDELETE flag in the appropriate property value. This allows multiple deletes and changes in a single call, which is much more efficient than multiple calls.

No memory is freed by the callee. Pointers in the CEPROPVAL structures can be anywhere in the caller's address space—they can be marshaled in like the array returned by CeReadRecordProps, or they can be independently allocated.

For more information, see Accessing Persistent Storage.

When writing applications for Windows CE version 1.0, use the PegWriteRecordProps function.

Detailed Description of Data Structures for a Database API

CHAPTER 95

Fsdbase Component: Structures

CEDBASEINFO

The CEDBASEINFO structure contains information about a database object. This structure is used by the CeSetDatabaseInfo and CeCreateDatabaseEx functions.

Syntax

```
typedef struct_CEDBASEINFO {
        DWORD     dwFlags
        WCHAR     szDbaseName
                  [CEDB_MAXDBASENAMELEN];
        DWORD     dwDbaseType;
        WORD      wNumRecords;
        WORD      wNumSortOrder;
        DWORD     dwSize;
        FILETIME  ftLastModified;
        SORTORDERSPEC
                  rgSortSpecs[CEDB_MAXSORTORDER];
} CEDBASEINFO
```

At a Glance

Header file:	Windbase.h
Platforms:	H/PC
Versions:	1.01 and later

Members

dwFlags

The LOWORD indicates the valid members of this structure. This member can be a combination of the following values:

CEDB_VALIDMODTIME
: The ftLastModified member is valid and should be used.

CEDB_VALIDNAME
: The szDbaseName member is valid and should be used.

CEDB_VALIDTYPE
: The dwDbaseType member is valid and should be used.

CEDB_VALIDSORTSPEC
: The rgSortSpecs member is valid and should be used.

CEDB_VALIDDBFLAGS
The LOWORD of dwFlags member is valid and should be used.

The HIGHWORD identifies the associated database properties. This member can be a combination of the following values:

CEDB_NOCOMPRESS
The database is not compressed. If this flag is used with CeSetDatabaseInfoEx, a compressed database is uncompressed. If this flag is used with CeCreateDatabaseEx, the database is not compressed.

To compress a database, CeSetDatabaseInfoEx or CeCreateDatabaseEx is called with CEDB_VALIDDBFLAGS and the HIGHWORD set to zero. By default, all databases are compressed. If you are going to change the compression, it should be done at creation time.

szDbaseName
Null-terminated string that contains the name of the database. The string can have up to 32 characters, including the termination null character. This member must be set when used for CeCreateDatabaseEx.

dwDbaseType
Type identifier for the database.

wNumRecords
Returns the number of records in the database.

wNumSortOrder
Number of sort orders active in the database. Up to four sort orders can be active at a time.

dwSize
Returns the size, in bytes, of the database.

ftLastModified
Returns the last time this database was modified.

rgSortSpecs
Array containing the sort order descriptions. Only the first *n* array members are valid, where *n* is the value specified by the wNumSortOrder member. If no sort orders are specified for CeCreateDatabaseEx or when CEDB_VALIDSORTSPEC is not specified, then a default sort order is assigned to the database.

See Also CeCreateDatabaseEx, CEOIDINFO, CeSetDatabaseInfoEx

CEOIDINFO

The CEOIDINFO structure contains information about an object in the object store.

Syntax

```
typedef struct_CEOIDINFO {
        WORD        wObjType;
        DWORD       dwSize;
        WORD        wPad;
        union {
                CEFILEINFO      infFile;
                CEDIRINFO       infDirectory;
                CEDBASEINFO     infDatabase;
                CERECORDINFO    infRecord;
        };
} CEOIDINFO;
```

At a Glance

Header file:	Windbase.h
Platforms:	H/PC
Versions:	1.01 and later

Members

wObjType

Type of the object. This member can be one of the following values:

OBJTYPE_INVALID
The object store contains no valid object that has this object identifier.

OBJTYPE_FILE
The object is a file.

OBJTYPE_DIRECTORY
The object is a directory.

OBJTYPE_DATABASE
The object is a database.

OBJTYPE_RECORD
The object is a record inside a database.

dwSize

Must be set to the size of CEOIDINFOEX, that is, size(CEOIDINFOEX).

wPad

Aligns the structure on a double-word boundary.

infFile
A CEFILEINFO structure that contains information about a file. This member is valid only if wObjType is OBJTYPE_FILE.

infDirectory
A CEDIRINFO structure that contains information about a directory. This member is valid only if wObjType is OBJTYPE_DIRECTORY.

infDatabase
A CEDBASEINFO structure that contains information about a database. This member is valid only if wObjType is OBJTYPE_DATABASE.

infRecord
A CERECORDINFO structure that contains information about a record in a database. This member is valid only if wObjType is OBJTYPE_RECORD.

See Also CEDBASEINFO, CEDIRINFO, CEFILEINFO, CERECORDINFO

CEPROPVAL

The CEPROPVAL structure contains a property value.

Syntax

```
typedef struct_CEPROPVAL {
    CEPROPID    propid;
    WORD        wLenData;
    WORD        wFlags;
    CEVALUNION  val;
} CEPROPVAL;
typedef CEPROPVAL *PCEPROPVAL;
```

At a Glance

Header file:	Windbase.h
Platforms:	H/PC
Versions:	1.01 and later

Members

propid
Identifier of the property value. The high-order word is an application-defined identifier, and the low-order word is a predefined constant value that indicates the data type of the value specified by the val member. The low-order word can be one of the following values:

CEVT_BLOB
A CEBLOB structure.

CEVT_FILENAME
A FILENAME structure.

CEVT_I2
A 16-bit signed integer.

CEVT_I4
A 32-bit signed integer.

CEVT_LPWSTR
A null-terminated string.

CEVT_UI2
A 16-bit unsigned integer.

CEVT_UI4
A 32-bit unsigned integer.

wLenData
Not used.

wFlags
Special flags for the property. This parameter can be one of the following values:

CEDB_PROPNOTFOUND
Set by the CeReadRecordProps function if the property was not found.

CEDB_PROPDELETE
If passed to the CeWriteRecordProps function, this flag causes the property to be deleted.

val
Actual value for simple types, or a pointer for strings or Binary Large Objects (BLOBs).

Remarks When writing applications for Windows CE version 1.0, use the PEGPROPVAL structure.

See Also CeReadRecordProps, CeSeekDatabase, CeWriteRecordProps

SORTORDERSPEC

The SORTORDERSPEC structure contains information about a sort order in a database.

Syntax

```
typedef struct_SORTORDERSPEC {
        PEGPROPID       propid;
        DWORD           dwFlags;
} SORTORDERSPEC;
```

At a Glance

Header file:	Windbase.h
Platforms:	H/PC
Versions:	1.0 and later

Members

propid

Specifies the identifier of the property to be sorted on. Sorts on binary properties are not allowed.

dwFlags

Specifies the sort flags. This parameter can be a combination of the following values:

CEDB_SORT_DESCENDING

The sort is done in descending order. By default, the sort is done in ascending order.

CEDB_SORT_CASEINSENSITIVE

The sort operation is case sensitive. This value is valid only for strings.

CEDB_SORT_UNKOWNFIRST

Records that do not contain this property are placed before all the other records. By default, such records are placed after all other records.

CEDB_SORT GENERICORDER

Remarks

The system supports only simple sorts on a primary key. Records with the same key value are sorted in arbitrary order.

See Also

CeCreateDatabase, CeDBASEINFO

Detailed Description of a Position and Navigation API

IPosNav

The IPosNav interface provides all the methods needed to utilize Apollo's GPS capabilities.

Method	Description
IPosNav::CloseHandle	Closes a P&N device
IPosNav::pnapiDeleteDeviceList	Deletes a linked list of PNDEVICE structures
IPosNav::pnapiFindDevices	Finds all connected P&N devices on the system
IPosNav::pnapiGetData	Retrieves various types of data from a P&N device
IPosNav::pnapiOpenDevice	Opens a P&N device for communication
IPosNav::pnapiSetData	Sends data to either the P&N device, or the registry
IPosNav::pnapiStartDirectCall	Starts a call to get data from the P&N device
IPosNav::pnapiStopDirectCall	Stops a IPosNav::pnapiStartDirectCall that has been started
IPosNav::pncnvBearingToVelocity	Converts a bearing and two speeds to East, North and Up velocities
IPosNav::pncnvDegreesToRadians	Converts latitude/longitude/altitude data from degrees to radians
IPosNav::pncnvPNTMToWintm	Converts time, in PNTM format, to Win32 SYSTEMTIME format
IPosNav::pncnvRandiansToDegrees	Converts latitude/longitude/altitude data from degrees to radians
IPosNav::pncnvVelocityToBearing	Converts North/East/Up velocity data to a bearing and two speeds
IPosNav::pncnvWintmToPNTM	Converts time in Win32 format to PNTM format

Remarks The Position and Navigation API (PNAPI) for the AutoPC is a subset of the full PNAPI. The IPosNav interface handles most GPS-related tasks. The other interface, IDGPS, contains a small set of methods that are needed to support differential GPS.

IPosNav::CloseHandle

The IPosNav::CloseHandle method is used to close a P&N device.

Syntax

```
HRESULT CloseHandle (
        hPNDevice hPN,
);
```

Parameters *hPN*
Handle to the P&N device to be closed.

Return Values S_OK
Function succeeded.
E_FAIL
Unspecified error.
E_INVALIDARG
One or more arguments are invalid.
E_NOTIMPL
Not implemented.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available (Unplugged? Dead?).
PNAPI_E_MEMFREE
Memory/resource cannot be freed.

Example XX

Remarks If this method is not called upon exiting, PNAPI resources will not be deleted.

This method must wait for pending calls to finish before stopping calls to a P&N device. It may therefore take a second or two to return.

See Also IPosNav::pnapiOpenDevice

IPosNav::pnapiDeleteDeviceList

The IPosNav::pnapiDeleteDeviceList method is used to delete a linked list of PNDEVICE structures

Syntax HRESULT pnapiDeleteDeviceList (
pPNDEVICE *pPNDeviceHead*
);

Parameters *pPNDeviceHead*
Pointer to the first structure in the linked list.

Return Values S_OK
Successful.
Errors
Returns the appropriate HRESULT error value.

Remarks After opening the selected P&N device(s), delete the PNDEVICE linked list by using the pnapiDeleteDeviceList function.

See Also IPosNav::pnapiFindDevices

IPosNav::pnapiFindDevices

The IPosNav::pnapiFindDevices method is used to find all connected pointing and navigation devices on the system.

Syntax

```
HRESULT pnapiFindDevices (
    ppPNDEVICE ppDevArray
    DWORD *pdwNumDev
);
```

Parameters

pDevArray
Pointer to an array of PNDEVICE pointers. Returns the head of a linked list of PNDEVICE structures. The user should destroy this list with the pnapiDeleteDeviceList function.

pdwNumDev
Returns the number of P&N devices found.

Return Values

S_OK
Function succeeded.

E_FAIL
Unspecified error.

E_INVALIDARG
One or more arguments are invalid.

E_NOTIMPL
Not implemented.

TYPE_E_DLLFUNCTIONNOTFOUND
Function not defined in specified DLL.

REGDB_E_READREGDB
Could not read key from registry.

PNAPI_E_INVALIDREGDBVALUE
Invalid value in registry.

PNAPI_E_REGDBCLOSEKEY
Can't close a registry key.

PNAPI_E_MEMFREE
Memory/resource cannot be freed.

PNAPI_E_BADOS
Invalid operating system version.

E_OUTOFMEMORY
PNAPI has run out of memory.

Remarks

The IPosNav::pnapiFindDevices method returns information for P&N devices in an 'unknown' status, but does not return data on a truly 'dead' P&N device.

See Also IPosNav::pnapiOpenDevice, IPosNav::pnapiDeleteDeviceList

IPosNav::pnapiGetData

The IPosNav::pnapiGetData method is used to get various types of data from a P&N device.

Syntax
```
HRESULT pnapiGetData (
    hPNDevice hPN,
    LPVOID pBuffer,
    DWORD dwSize,
    PNData_t DataType
);
```

Parameters

hPN

P&N handle for the P&N device to use.

pBuffer

Pointer to the buffer that will receive the data. If any part of the requested data cannot be found, the corresponding entry in the PNAV structure that will be part of the buffer is marked as invalid.

dwSize

Size of *pBuffer*.

DataType

Type of data to get from the P&N device. The following types of data can be requested.

Data Type	Description	Structure Type
PN_DT_POSITION	Long,lat,alt position data	PNPOSITION
PN_DT_VELOCITY	Velocity data	PNVELOCITY
PN_DT_DEVICESTATE	Device state data	PNDEVSTATE
PN_DT_TIME	Time data	PNTIME
PN_DT_TM	Time data	PNTM
PN_DT_ACCURACY	Accuracy data	PNACCURACY
PN_DT_STATION	Station data	PNSTATION
PN_DT_DEVICE	Device profile data	PNDEVICE
PN_DT_CONFIG	Configuration data	PNCONFIG
PN_DT_SETTINGS	Settings data	PNSETTINGS
PN_ST_DGPSSTATUS	Differential GPS status data	PNDGPSSTATUS
PN_DT_ALMANAC	Almanac data	PNALMANAC

Return Values S_OK

Function succeeded.

E_FAIL

Unspecified error.

E_INVALIDARG

One or more arguments are invalid.

E_NOTIMPL
Not implemented.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available.
PNAPI_E_STRUCTLOCKED
Data structure is locked.
PNAPI_E_NOCALLSTARTED
No call has been started yet.
PNAPI_E_NODATAYET
No data has been received from the P&N device yet.

Remarks PNAPI allows various OEM defined PNData_t structures to be passed through this function so that specific features can be made available. The quantity of available calls can be found within the header file included with this document. These calls start at PN_DT_START_c and end at PN_DT_END_c. OEM vendors should provide details about how they have implemented these OEM defined PNData_t's.

All data is received from the P&N device *except* PNCONFIG data which is taken from the registry.

The almanac data is GPS specific and provides knowledge of the position of the satellites in the sky.

See Also IPosNav::pnapiSetData, IPosNav::pnapiStartDirectCall

IPosNav::pnapiOpenDevice

The IPosNav::pnapiOpenDevice method is used to open communication with a GPS device.

Syntax
```
HRESULT pnapiOpenDevice (
    phPNDevice phPN,
    pPNDEVICE pDevice
);
```

Parameters *phPN*
Handle to a Pointing and Navigation device (phPNDevice is declared as LPVOID). If successful, a valid P&N handle is returned via this parameter.
pDevice
Pointer to the PNDEVICE profile structure for the device to be opened. This structure is returned by pnapiFindDevices.

Return Values S_OK
Function succeeded.

E_FAIL
Unspecified error.
E_INVALIDARG
One or more arguments are invalid.
E_NOTIMPL
Not implemented.
E_OUTOFMEMORY
Ran out of memory.
REGDB_E_READREGDB
Could not read key from registry.
PNAPI_E_REGDBCLOSEKEY
Can't close a registry key.
PNAPI_E_LOADDLL
Can't load DLL.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available.

Remarks PNAPI allows multiple applications to use a P&N device simultaneously. An application should first use pnapiFindDevices to locate the device. When the first application opens a P&N device, PNAPI initializes the P&N device according to the control panel settings (initializing a rough position and time). When a second application opens the same P&N device, PNAPI does not initialize the P&N device a second time.

Close the P&N device using the CloseHandle function.

See Also IPosNav::pnapiFindDevices, IPosNav::CloseHandle

IPosNav::pnapiSetData

The IPosNav::pnapiSetData method is used to send data to either the P&N device, or the registry.

Syntax
```
HRESULT pnapiSetData (
    hPNDevice hPN,
    LPVOID pBuffer,
    DWORD dwSize,
    PNData_t Data_Type
);
```

Parameters
hPN
Handle for the P&N device to use.
pBuffer
Pointer to a buffer to hold the data. The format is determined by *Data_Type*.
dwSize
Size of *pBuffer*, in bytes.

Data_Type

Type of data to set. The supported data types are:

Data Type	Description	Structure Type
PN_DT_POSITION	Long,lat,alt position data	PNPOSITION
PN_DT_VELOCITY	Velocity data	PNVELOCITY
PN_DT_DEVICESTATE	Device state data	PNDEVSTATE
PN_DT_TIME	Time data	PNTIME
PN_DT_TM	Time data	PNTM
PN_DT_ACCURACY	Accuracy data	PNACCURACY
PN_DT_STATION	Station data	PNSTATION
PN_DT_DEVICE	Device profile data	PNDEVICE
PN_DT_CONFIG	Configuration data	PNCONFIG
PN_DT_SETTINGS	Settings data	PNSETTINGS
PN_DT_DGPSSTATUS	Differential GPS status data	PNDGPSSTATUS
PN_DT_ALMANAC	Almanac data	PNALMANAC

Return Values

Return Value	Meaning
S_OK	Function succeeded.
E_FAIL	Unspecified error.
E_INVALIDARG	One or more arguments are invalid.
E_NOTIMPL	Not implemented.
PNAPI_E_DEVICEUNAVAILABLE	P&N device not available.
PNAPI_E_NOACCESS	Application has insufficient access rights.

Remarks

The position, time can be set to allow the P&N device to find its position more quickly.

The configuration data in the PNCONFIG structure will be stored in the registry by this function. The settings contained will also be used to update the configuration of the P&N device. If any parameters do not apply to the P&N device, then they will be ignored by PNAPI.

Almanac data is GPS specific and is received from the P&N device by the IPosNav::pnapiGetData or IPosNav::pnapiStartDirectCall function. The almanac details are stored in the registry only through the PNCONFIG structure. The almanac data should not be altered in any way. It provides accurate information about the GPS satellites' position at any one time. If almanac data is passed to this function, the system may be able to get a fix faster.

PNAPI allows various OEM defined PNData_t objects (structures, usually) to be passed through this function so that specific features can be made available. The quantity of available calls can be found within the header file included with this document. These calls start at PN_DT_START_c and end at PN_DT_END_c. OEM vendors should provide details about how they have implemented these OEM defined PNData_t's.

All data is sent to the P&N device *except* PNCONFIG data which is sent to the registry.

Only applications with READ/WRITE access can use this function - the exception being when the user wishes to change access rights.

The PNTIME structure should contain a fairly accurate time in UTC (Universal Coordinated Time - also known as Greenwich mean time).

See Also IPosNav::pnapiGetData, IPosNav::pnapiStartDirectCall

IPosNav::pnapiStartCall

The IPosNav::pnapiStartCall method starts a call to get data from the P&N device and place it in PNAPI data structures.

Syntax

```
HRESULT pnapiStartCall (
        hPNDevice hPN,
        PNData_t Call,
        DWORD dwPeriod,
);
```

Parameters

hPN

The P&N device handle.

Call

Type of call to get from P&N device. All PNData_t calls valid for the pnapiGetData function can be used for *Call*.

dwPeriod

Time period between updates of data, in milliseconds. If *dwPeriod*=0, only one call will be made. If *dwPeriod*=1, the call can be made as rapidly as the device permits.

Return Values S_OK

Function succeeded.

E_FAIL

Unspecified error.

E_INVALIDARG
One or more arguments are invalid.
N_NOTIMPL
Not implemented.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available.
PNAPI_E_DATAUNAVAILABLE
Data unavailable.
PNAPI_S_CALLALREADYSTARTED
(Warning) Call already started.
PNAPI_S_PERIODTOOSMALL
(Warning) P&N device unable to support a call period as fast as that being requested.

Remarks This method instructs the device to update its associated data structures at specified intervals. It enables a user to get the most recent data using the pnapiGetData method from the P&N device's data structures within PNAPI as often as needed.

See Also IPosNav::pnapiStopCall, IPosNav::pnapiGetData

IPosNav::pnapiStartDirectCall

The IPosNav::pnapiStartDirectCall method starts a call to get data from the P&N device.

Syntax

```
HRESULT pnapiStartDirectCall (
    hPNDevice hPN,
    PNData_t Call,
    DWORD dwPeriod,
    HWND hWnd
);
```

Parameters

hPN
The P&N device handle.
Call
Type of call to get from P&N device. All PNData_t calls valid for the pnapiGetData function can be used for *Call*.
dwPeriod
Time period between updates of data, in milliseconds.
hWnd
The *HWND* that will receive messages informing the user that the data has been updated, and receive the data.

Return Values S_OK
Function succeeded.

E_FAIL
Unspecified error.
E_INVALIDARG
One or more arguments are invalid.
E_NOTIMPL
Not implemented.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available (Unplugged? Dead?).
PNAPI_S_CALLALREADYSTARTED
(Warning) Call already started.
PNAPI_S_PERIODTOOSMALL
(Warning) P&N device unable to support a call period as fast as that being requested.

Remarks

Like pnapiGetData, this method allows the OEM defined PNData_t's to be used. For more information, see the pnapiGetData method. All data is received from the P&N device *except* PNCONFIG data which is taken from the registry.

This method will get the requested data every *dwPeriod*, and then post a message to the owner window. The time between updates, *dwPeriod*, is in milliseconds, so presently calls of a period of >2 weeks can be made. If *dwPeriod*=0 then only one call will be made. If *dwPeriod*=1 then the call will be made as rapidly as the P&N device will allow. OEMs should specify in their documentation the maximum and minimum periods that their P&N devices support.

When data is received from the P&N device, PNAPI posts a WM_COPYDATA message. The LPARAM parameter contains a COPYDATASTRUCT structure which contains two parameters – *dwData* and *lpData*. *dwData* specifies the type of data being passed. *lpData* is a pointer to the relevant structure cast to an LPVOID. See WM_COPYDATA notes in Win32 help for more information.

UINT	dwData	lpData	Meaning
WM_COPYDATA	PN_DT_POSITION	Pointer to PNPOSITION data	PNPOSITION data has been returned
WM_COPYDATA	PN_DT_VELOCITY	Pointer to PNVELOCITY data	PNVELOCITY data has been returned
WM_COPYDATA	PN_DT_TIME	Pointer to PNTIME data	PNTIME data has been returned
WM_COPYDATA	PN_DT_DEVICESTATE	Pointer to PNDEVSTATE data	PNDEVSTATE data has been returned

WM_COPYDATA	PN_DT_ACCURACY	Pointer to PNACCURACY data	PNACCURACY data has been returned
WM_COPYDATA	PN_DT_STATION	Pointer to PNSTATION data	PNSTATION data has been returned
WM_COPYDATA	PN_DT_CONFIG	Pointer to PNCONFIG data	PNCONFIG data has been returned
WM_COPYDATA	PN_DT_ALMANAC	Pointer to PNALMANAC data	PNALMANAC data has been returned
WM_COPYDATA	PN_DT_SETTINGS	Pointer to PNSETTINGS data	PNSETTINGS data has been returned

See Also IPosNav::pnapiStopDirectCall, IPosNav::pnapiGetData

IPosNav::pnapiStopCall

The IPosNav::pnapiStopCall method is used to stop a IPosNav::pnapiStartCall that has been started.

Syntax
```
HRESULT pnapiStopCall (
    hPNDevice hPN,
    PNData_t Call
);
```

Parameters

hPN

The P&N device handle.

Call

Type of call to stop. All calls that are valid for the IPosNav::pnapiStartCall function are valid for the IPosNav::pnapiStopCall function.

Return Values

S_OK

Function succeeded.

E_FAIL

Unspecified error.

E_INVALIDARG

One or more arguments are invalid.

E_NOTIMPL

Not implemented.

PNAPI_E_DEVICEUNAVAILABLE

P&N device not available (Unplugged? Dead?).

PNAPI_E_NOCALLSTARTED
No call has been started yet.

Remarks If a call has been started (using IPosNav::pnapiStartCall) with a period of 0, then it does not need to be stopped with IPosNav::pnapiStopCall. A period of 0 indicates that the call is made only once, and then it is automatically stopped.

See Also IPosNav::pnapiStartCall

IPosNav::pnapiStopDirectCall

The IPosNav::pnapiStopDirectCall method is used to stop a IPosNav::pnapiStartDirectCall that has been started.

Syntax
```
HRESULT pnapiStopDirectCall (
    hPNDevice hPN,
    PNData_t Call
);
```

Parameters *hPN*
The P&N device handle.
Call
Type of call to stop. All calls that are valid for the IPosNav::pnapiStartDirectCall function are valid for the IPosNav::pnapiStopDirectCall function.

Return Values S_OK
Function succeeded.
E_FAIL
Unspecified error.
E_INVALIDARG
One or more arguments are invalid.
E_NOTIMPL
Not implemented.
PNAPI_E_DEVICEUNAVAILABLE
P&N device not available (Unplugged? Dead?).
PNAPI_E_NOCALLSTARTED
No call has yet been started.

Remarks If a call has been started (using IPosNav::pnapiStartDirectCall) with a period of 0, then this call does not need to be stopped with IPosNav::pnapiStopDirectCall. A period of 0 indicates that the call is made only once, and then is automatically stopped.

See Also IPosNav::pnapiStartDirectCall

IPosNav::pncnvBearingToVelocity

The IPosNav::pncnvBearingToVelocity method is used to convert a bearing and two speeds to East, North and Up velocities.

Syntax

```
HRESULT pncnvVelocityToBearing (
    const pPNVELENU pENUVel,
    pPNVELBEAR pBearVel,
);
```

Parameters

pENUVel
Pointer to a PNVELENU structure holding the velocity data.

pBearVel
Pointer to a PNVELBEAR structure holding the bearing data.

See Also IPosNav::pncnvVelocityToBearing, PNVELENU, PNVELBEAR

IPosNav::pncnvDegreesToRadians

The IPosNav::pncnvDegreesToRadians method is used to convert latitude/longitude/altitude data from degrees to radians.

Syntax

```
HRESULT pncnvDegreesToRadians (
    pPNPOSLLA pLLAPos
);
```

Parameters

pLLAPos
Pointer to a PNPOSLLA structure containing the latitude/longitude/altitude data. The structure is returned with the same position in radians.

Return Values S_OK
Function succeeded.

E_INVALIDARG
One or more arguments are invalid.

See Also IPosNav::pncnvRadiansToDegrees, PNPOSLLA

IPosNav::pncnvPNTMToWintm

The IPosNav::pncnvPNTMToWintm method is used to convert time, in PNTM format, to Win32 SYSTEMTIME format.

Syntax
```
HRESULT pncnvPNTMToWintm (
    PNTM pNTM,
    const SYSTEMTIME pTime,
);
```

Parameters

pNTM
The time to be converted, in PNTM format.

pTime
Receives the returned Win32 SYSTEMTIME formatted time.

Return Values

S_OK
Function succeeded.

E_FAIL
Unspecified error.

E_INVALIDARG
One or more arguments are invalid.

See Also IPosNav::pncnvWintmToPNTM, PNTM

IPosNav::pncnvRadiansToDegrees

The IPosNav::pncnvRadiansToDegrees method is used to convert latitude/longitude/altitude data from radians to degrees.

Syntax
```
HRESULT pncnvRadiansToDegrees (
    pPNPOSLLA pLLAPos
);
```

Parameters

pPLLAPos

Pointer to a PNPOSLLA structure containing the latitude/longitude/altitude data. The structure is returned with the same position in degrees.

Return Values

S_OK
Function succeeded.

E_INVALIDARG
One or more arguments are invalid.

See also. IPosNav::pncnvDegreesToRadians, PNPOSLLA

IPosNav::pncnvVelocityToBearing

The IPosNav::pncnvVelocityToBearing method is used to convert North/East/Up velocity data to a bearing and two speeds.

Syntax

```
HRESULT pncnvVelocityToBearing (
    pPNVELBEAR pBearVel,
    const pPNVELENU pENUVel,
);
```

Parameters

pBearVel
Pointer to a PNVELBEAR structure to hold the bearing data.

pENUVel
Pointer to a PNVELENU structure holding the velocity data.

Return values

S_OK
Function succeeded.

E_INVALIDARG
One or more arguments are invalid.

See Also IPosNav::pncnvBearingToVelocity, PNVELENU

IPosNav::pncnvWintmToPNTM

The IPosNav::pncnvWintmPNTM method is used to convert time in Win32 format to PNTM format.

Syntax

```
HRESULT pncnvWintmPNTM (
    const SYSTEMTIME pTime,
    PNTM pNTM,
);
```

Parameters

pTime
The time to be converted, in Win32 SYSTEMTIME format.

pNTM
Receives the returned PNTM formatted time.

Return values

S_OK
Function succeeded.

E_FAIL
Unspecified error.

E_INVALIDARG
One or more arguments are invalid.

See Also IPosNav::pncnvPNTMToWintm

IDGPS

The IDGPS interface provides methods to handle differential GPS devices.

Method	Description
IDGPS::Close	Closes a DGPS device
IDGPS::GetRTCM	Gets an RTCM message from a DGPS device
IDGPS::GetServiceQuality	Gets the DGPS service quality
IDGPS::Open	Opens a DGPS device

Remarks The IDGPS interface contains a smaller set of methods that are needed to support differential GPS.

Because of the variety of ways DGPS can be handled, this SDK only provides a definition of the IDGPS interface, not an implementation. To utilize DGPS, developers must create an object which exposes the IDGPS interface, along with whatever code is necessary for such tasks as managing communication with the base station. The details of the IDGPS implementation will depend on the specifics of the particular DGPS system.

See Also IPosNav

IDGPS::Close

The IDGPS::Close method is used to close a DGPS device.

Syntax HRESULT Close (void);

Parameters None

Return Values S_OK
Method succeeded.
E_FAIL
Method failed.

See Also IDGPS::Open

IDGPS::GetRTCM

The IDGPS::GetRTCM method gets a Radio Technical Commission for Maritime Service (RTCM) message from the DGPS device.

Syntax

```
HRESULT GetRTCM (
    DWORD   dwMessageID
    PVOID   pData
    DWORD   dwSize
);
```

Parameters

dwMessageID
The RTCM message number (in).

pData
Pointer to a buffer to store the returned RTCM message (out).

dwSize
The size of the structure being passed (out).

Return Values

S_OK
Method failed.

E_FAIL
Unspecified error.

IDGPS::GetServiceQuality

The IDGPS::GetServiceQuality method is used to determine the quality of support this DGPS service can provide.

Syntax

```
HRESULT GetServiceQuality (
    DWORD &rdwMessage
    DWORD &rdwUpdateRate
);
```

Parameters

rdwMessage
Holds the DGPS service quality. *rdwUpdateRate*

Holds the fastest rate that this DGPS service can hope to update its fastest RTCM message.

Return Values

S_OK
Method succeeded.

E_FAIL
Method failed.

IDGPS::Open

The IDGPS::Open method is used to open a DGPS device.

Syntax HRESULT Open (void);

Parameters None

Return Values S_OK
Method succeeded.
E_FAIL
Method failed.

See Also IDGPS::Close

Detailed Description of Data Structures for a Position and Navigation API

CHAPTER 19

PN3State _t

Enumerates a set of available modes.

Constant	Value	Description
PN_3S_FALSE	0	Off, or FALSE position
PN_3S_TRUE	1	On, or TRUE position
PN_3S_OTHER	2	Other, or indeterminate position

PNAccess_t

Enumerates the access rights that the P&N device can supply to the application.

Constant	Value	Meaning
PN_AS_READWRITE	MIN_PNACCESS_T	P&N device has full access rights
PN_AS_READ	MAX_PNACCESS_T	P&N device has partial access rights (allows user to only receive data from the P&N device).

PNACCURACY

Stores accuracy details about the position supplied by the P&N device and the time these details were last updated.

```
typedef struct tagPNACCURACY
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    PNDouble        dHorizError;
    PNDouble        dVerticalError;
    PNDouble        dEDOP;
    PNDouble        dNDOP;
    PNDouble        dVDOP;
    PNDouble        dPDOP;
    PNDouble        dTDOP;
    PNDouble        dGDOP;
    PNAVACCURACY    acAvAccuracy;
    DWORD           dwPNReserved;
```

```
} PNACCURACY;
```

Members

dwStructureSize
The size, in bytes, of the structure.

tiTime
The time the data was received.

dHorizError
Not used by Windows CE.

dVerticalError
Not used by Windows CE.

dEDOP
East dilution of precision.

dNDOP
North dilution of precision.

dVDOP
Vertical dilution of precision.

dPDOP
Position dilution of precision.

dTDOP
Time dilution of precision.

dGDOP
Geometric dilution of precision.

acAvAccuracy
Stores which elements of acAvAccuracy of are valid and which are not.

dwPNReserved
Reserved for future use by PNAPI.

PNALMANAC

Stores GPS almanac details.

```
typedef struct tagPNALMANAC
{
    DWORD       dwStructureSize;
    PNTIME      tiTime;
    PNSATELLITE saSatellite (PN_NUM_SATS_c);
    DWORD       dwPNReserved;
} PNALMANAC;
```

Members

dwStructureSize
The size, in bytes, of the structure.

tiTime
Time data was collected.

saSatellite
Satellite information.

dwPNReserved

Reserved for future use by PNAPI.

Remarks The index number for each PNSATELLITE structure is PRN#/SVID of the satellite in question. However, as the index number goes from 0-31, the index number+1 = PRN#/SVID.

tiTime stores the time this almanac data was collected. To be precise, it is the time the first piece of satellite information was received.

PNAVACCURACY

Stores which PNACCURACY elements are valid and which are not.

```
typedef struct tagPNAVACCURACY
{
        DWORD     dwStructureSize
        DWORD     dwAvl;
        DWORD     dwPNReserved;
} PNAVACCURACY
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The dwAvl parameter contains bit flags – one for each element in the corresponding PNACCURACY structure that shows whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_AAC_AHORIZERROR	0	Not used by Windows CE.
PN_AAC_AVERTICALERROR	1	Not used by Windows CE.
PN_AAC_EDOP	2	EDOP valid / invalid.
PN_AAC_NDOP	3	NDOP valid / invalid.
PN_AAC_VDOP	4	VDOP valid / invalid.
PN_AAC_PDOP	5	PDOP valid / invalid.
PN_AAC_TDOP	6	TDOP valid / invalid.
PN_AAC_GDOP	7	GDOP valid / invalid.
Reserved for future use.	8-31	

dwPNReserved

Reserved for future use by PNAPI.

PNAVDEVSTATE

Stores which DEVSTATE elements are valid and which are not.

```
typedef struct tagPNAVDEVSTATE
{
        DWORD     dwStructureSize;
        DWORD     dwAvl;
        DWORD     wPNReserved
} PNAVDEVSTATE;
```

Members

dwStructureSize
The size, in bytes, of the structure.

dwAvl
The dwAvl parameter contains bit flags – one for each element in the corresponding PNDEVSTATE structure that shows whether the element is available. The following bit flag is defined for this structure:

Name	Bit Flag	Meaning
PN_ADS_STATE	0	Device state valid / invalid.
Reserved for future use	1-31	

dwPNReserved
Reserved for future use by PNAPI.

PNAVDGPSSTATUS

Holds status information for differential GPS.

```
typedef struct tagPNAVDGPSSTATUS
{
        DWORD     dwStructureSize;
        DWORD     dwAvl;
        DWORD     dwPNReserved;
} PPNAVDGPSSTATUS;
```

Members

dwStructureSize
The size, in bytes, of the structure.

dwAvl
TBD.

dwPNReserved
Reserved.

PNAVINDSTATION

Shows which PNINDSTATION elements are valid and which are not.

```
typedef struct tagPNAVINDSTATION
{
    DWORD    dwStructureSize;
    DWORD    dwAvl;
    DWORD    dwPNReserved;
} PNAVINDSTATION;
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The dwAvl parameter contains bit flags – one for each element in the corresponding PNINDSTATION structure. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_ASI_STATE	0	Station state valid / invalid.
PN_ASI_STATIONIDNUM	1	Station ID number valid / invalid.
PN_ASI_USED	2	fUsed parameter valid / invalid.
PN_ASI_ELEVATION	3	Satellite elevation valid / invalid.
PN_ASI_SATAZIMUTH	4	Satellite azimuth valid / invalid.
PN_ASI_SIGNALSTRENGTH	5	Signal strength valid / invalid.
PN_ASI_COVERAGE	6	Not used by Windows CE.
Reserved for future use.	7-31	

dwPNReserved

Reserved for future use by PNAPI.

PNAVPOSLLA

Shows which of the position elements are valid. It is intended to mirror PNPOSLLA structure.

```
typedef struct tagPNAVPOSLLA
{
    DWORD    dwStructureSize;
    DWORD    dwAvl;
    DWORD    dwPNReserved;
} PNAVPOSLLA;
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The *dwAvl* parameter contains bit flags – one for each element in the corresponding PNPOSLLA structure that shows whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_APL_LONG	0	Longitude valid / invalid.
PN_APL_LAT	1	Latitude valid / invalid.
PN_APL_ALT	2	Altitude valid / invalid.
PN_APL_RADIANS	3	fRadians parameter valid / invalid.
Reserved for future use.	4-31	

dwPNReserved

Reserved for future use by PNAPI.

PNAVSATELLITE

Shows which PNSATELLITE elements are valid and which are not.

```
typedef struct tagPNAVSATELLITE
{
    DWORD    dwStructureSize;
    DWORD    dwAvl;
    DWORD    dwPNReserved;
} PNAVSATELLITE;
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The *dwAvl* parameter contains bit flags – one for each element in the corresponding PNSATELLITE structure that shows whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_ASA_SETDATA	0	Not used by Windows CE.
PN_ASA_PRN	1	PRN# valid / invalid.
PN_ASA_SATHEALTH	2	Satellite heath valid / invalid.
PN_ASA_REFWEEKNUMBER	3	Reference week number valid / invalid.
PN_ASA_REFTIMEOFWEEK	4	Referenced time of week valid / invalid.
PN_ASA_ECCENTRICITY	5	Eccentricity valid / invalid.

PN_ASA_ROOTSEMIMAJORAXIS	6	Square root semi-major axis valid / invalid.
PN_ASA_ARGUMENTOFPERIGEE	7	Argument of perigee valid / invalid.
PN_ASA_MEANANOMALYATREFTIME	8	Mean anomaly at reference time valid / invalid.
PN_ASA_RIGHTASCENSIONATREFTIME	9	Right ascension at reference time valid / invalid.
PN_ASA_RATERIGHTASCENSION	10	Rate of right ascension valid / invalid.
PN_ASA_CORRECTTOINCLINATION	11	Correction to inclination valid / invalid.
PN_ASA_AF0CLOCKCORRECT	12	AF0 clock correction valid / invalid
PN_ASA_AF1CLOCKCORRECT	13	AF1 clock correction valid / invalid.
Reserve for future use.	14-31	

dwPNReserved
Reserved for future use by PNAPI.

PNAVSETTINGS

Shows which PNSETTINGS elements are valid and which are not.

```
typedef struct tagPNAVSETTINGS
{
    DWORD    dwStructureSize;
    DWORD    dwAvl;
    DWORD    dwPNReserved;
} PNAVSETTINGS;
```

Members

dwStructureSize
The size, in bytes, of the structure.

dwAvl
The *dwAvl* parameter contains bit flags – one for each element in the corresponding PNSETTINGS structure that shows whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_ASE_MODE	0	Not used by Windows CE.
PN_ASE_DGPSENABLE	1	Enable differential GPS.
PN_ASE_DRENABLE	2	Enable dead reckoning.
PN_ASE_DGPSTIMEOUT	3	DGPS timeout.
PN_ASE_DGPS2DENABLE	4	Not used by Windows CE.

PN_ASE_DGPS2DTIMEOUT	5	Not used by Windows CE.
PN_ASE_DATUM	6	Datum valid / invalid.
PN_ASE_POWERSTATE	7	Power state valid / invalid.
PN_ASE_ALTITUDEHOLD	8	Not used by Windows CE.
PN_ASE_AHALTITUDE	9	Not used by Windows CE.
PN_ASE_2DPOSMODE	10	Not used by Windows CE.
PN_ASE_2DALTITUDE	11	Not used by Windows CE.
PN_ASE_ENVIRONMENT	12	Environment valid / invalid.
PN_ASE_ACCESS	13	Access rights valid / invalid.
Reserved for future use.	14-31	

dwPNReserved
Reserved for future use by PNAPI.

PNAVSTATION

Shows which PNSTATION elements are valid and which are not.

```
typedef struct tagPNAVSTATION
{
    DWORD    dwStructureSize;
    DWORD    dwAvl;
    DWORD    dwPNReserved;
} PNAVSTATION;
```

Members dwStructureSize
The size, in bytes, of the structure.

dwAvl
The dwAvl parameter contains bit flags – one for each element in the corresponding PNSTATION structure that shows whether the element is available. The following bit flags are defined for this structure.

Name	Bit Flag	Meaning
PN_ASN_NUMAVAILABLE	0	Not used by Windows CE.
PN_ASN_NUMUSED	1	Number stations used valid / invalid.
Reserved for future use.	2-31	

dwPNReserved
Reserved for future by PNAPI.

PNAVTM

Stores which PNTM elements are valid and which are not.

Syntax

```
typedef struct tagPNAVTM
{
        DWORD    dwStructureSize;
        DWORD    dwAvl;
        DWORD    dwPNReserved;
} PNAVTM;
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The *dwAvl* parameter contains bit flags – one for each element in the corresponding PNTM structure that shows whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_ATM_MILLISEC	0	Millisecond valid / invalid.
PN_ATM_DAY	1	Day valid / invalid.
Reserved for future use.	2-31	

dwPNReserved

Reserved for future use by PNAPI.

PNAVVELENU

Shows which velocity elements are valid and which are not.

```
typedef struct tagPNAVVELENU
{
        DWORD    dwStructureSize;
        DWORD    dwAvl;
        DWORD    dwPNReserved;
} PNAVVELENU;
```

Members

dwStructureSize

The size, in bytes, of the structure.

dwAvl

The dwAvl parameter contains bit flags – one for each element in the corresponding PNVELENU structure. They show whether the element is available. The following bit flags are defined for this structure:

Name	Bit Flag	Meaning
PN_AVN_EAST	0	East velocity valid / invalid.
PN_AVN_NORTH	1	North velocity valid / invalid.
PN_AVN_UP	2	Up velocity valid / invalid.
Reserved for future use.	3-31	

dwPNReserved
Reserved for future use.

PNCONFIG

Stores the data that goes into the registry as saved configuration data for this P&N device.

```
typedef struct tagPNCONFIG
{
    DWORD       dwStructureSize;
    PNPOSITION  poPositionData;
    PNACCURACY  acAccuracy;
    PNPOSITION  poStaticRefPos;
    PNALMANAC   alAlmanac;
    PNSETTINGS  seSettings;
    PNBool      fInitAlmanac;
    PNBool      fInitPosition;
    PNBool      fInitTime;
    DWORD       dwPNReserved;
} PNCONFIG;
```

Members

dwStructureSize
The size, in bytes, of the structure.

poPositionData
Holds position and time it was found. Only PNPOSLLA portion used by Windows CE.

acAccuracy
Not used by Windows CE.

poStaticRefPos
Not used by Windows CE.

alAlmanac
Almanac data.

seSettings
Not used by Windows CE.

fInitAlmanac
Whether almanac will be initialized on start up.

fInitPosition
Whether position will be initialized on start up.

fInitTime
Whether the time will be initialized on start up.

dwPNReserved
Reserved for future use by PNAPI.

Remarks All position data stored in these structures is stored in Longitude, Latitude, Altitude format in radians. If any structure contains a tiTime parameter, it shows when the data was gathered.

Note: all values in the PNCONFIG structure go to the registry. No information is passed to the device.

PNData_t

PNdata_t enumerates the types of data to be used by functions such as pnapiGetData and pnapiSetData.

Data Type	Description
PN_DT_ALL	All PNData_ts fields.
PN_DT_POSITION	Longitude, latitude, altitude position data (PNPOSLLA format).
PN_DT_VELOCITY	Velocity data (PNVELOCITY format).
PN_DT_DEVICESTATE	Device state data (PNDEVSTATE format).
PN_DT_TIME	Time data (PNTIME format).
PN_DT_TM	Time data (PNTM format).
PN_DT_ACCURACY	Accuracy data (PNACCURACY format).
PN_DT_STATION	Station data (PNSTATION format).
PN_DT_DEVICE	Device profile data (PNDEVICE format).
PN_DT_CONFIG	Configuration data (PNCONFIG format).
PN_DT_SETTINGS	Settings data (PNSETTINGS format).
PN_DT_STATICREFPOS	Not used by Windows CE.
PN_DT_DGPSSTATUS	Diff GPS status data (PNDGPSSTATUS format).
PN_DT_RTCM1	Not used by Windows CE.
PN_DT_ALMANAC	Almanac data (PNALMANAC format).
PN_DT_STATUS	Not used by Windows CE.
PN_DT_RESET	Not used by Windows CE.

PNDatum_t

Enumerates the links between datum and datum code.

Constant	Value	Meaning
PN_DA_WGS84	0	World Geodetic System 1984

Remarks Only WGS84 is valid.

PNDEVICE

The PNDEVICE structure contains a profile of a GPS device. In the case of multiple devices, the last element in the structure is a pointer to another PNDEVICE structure, and can be used to form a linked list of structures.

```
typedef struct tagPNDEVICE
{
    DWORD       dwStructureSize;
    WCHAR       szManufacturer [PN_MNFCT_SIZE_c];
    WCHAR       szModel [PN_MODEL_SIZE_c];
    PNReceiver_trtReceiverType;
    DWORD       dwUseCount;
    DWORD       dwQuality;
    WCHAR       szComPort [PN_COM_PORT_LEN_c];
    WCHAR       szRegRoot [PN_REG_PATH_LEN_c];
    DWORD       dwComPort;
    DWORD       dwPNReserved;
    struct tagPNDEVICE* pNext;
} PNDEVICE;
```

Members

dwStructureSize
The size, in bytes, of the structure.

szManufacturer
Not used by Windows CE.

szModel
The GPS chip manufacture and model name.

rtReceiverType
Not used by Windows CE.

dwUseCount
Number of applications that are currently using this device.

dwQuality
Quality of data this device can deliver (the lower the number the better it is).

100
Highest quality service. Supports all PNAPI features.

200
Rockwell/Trimble binary standard. Supports most PNAPI features.

300
Garmin standard. Supports not quite as many features as 200.
400
NMEA V2.1 standard. Supports some features.
500
NMEA V2.0 / V1.5 standard.
600
NMEA V1.0 standard.
700
Will support basic position and not much else.
800
Will give position, but not necessarily altitude.
900
Very basic support.

szComPort
Not used by Windows CE.

szRegRoot
For PNAPI internal use.

pNext
For multiple devices, pNext points to the next structure in a linked list.

dwComPort
COM port in numerical format (see PN_I2P_GPS1_c and PN_I2P_GPS2Pc).

dwPNReserved
Reserved for future use by PNAPI.

PNDeviceState

Enumerates the possible device states.

State	Value	Description
PN_DS_INVALIDDS	-1000	//Device State is in invalid state.
PN_DS_NOTPRESENT	MIN_DEVICESTATE_T	//Device not present (i.e. been unplugged)
PN_DS_ERROR	1	//Error in device making it not operate at all.
PN_DS_WARNING	2	//Error with device but can still operate.
PN_DS_OK	3	//Device 100% OK (but not yet searching).
PN_DS_SEARCHING	4	//Searching for fix.

PN_DS_LEVEL1	5	//Found level 1 accuracy data.
PN_DS_LEVEL2	6	//Found level 2 accuracy data.
PN_DS_LEVEL3	7	//Found level 3 accuracy data.
PN_DS_LEVEL4	8	//Found level 4 accuracy data.
PN_DS_LEVEL5	9	//Found level 5 accuracy data.
PN_DS_LEVEL6	10	//Found level 6 accuracy data.
PN_DS_FOUND1SAT	11	//Found 1 satellite (GPS specific).
PN_DS_FOUND2SATS	12	//Found 2 satellites (GPS specific).
PN_DS_NOTIME	MAX_DEVICESTATE_T	//No GPS time found (GPS specific).

PNDEVSTATE

Stores the P&N device state and what time it was last updated.

```
typedef struct tagPNDEVSTATE
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    PNDeviceState_t dsState;
    PNAVDEVSTATE    dsAvState;
    DWORD           dwPNReserved;
} PNDEVSTATE;
```

Members

dwStructureSize
: The size, in bytes, of the structure.

tiTime
: The time of the last update.

dsState
: The device state.

dsAvState
: Shows which dsState elements are valid and which are not.

dwPNReserved
: Reserved for future use.

PNDGPSSTATUS

Holds the differential GPS status.

```
typedef struct tagPNDGPSSTATUS
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    PN3State_t      DGPSMode;
    PN3State_t      OperatingMode;
    PNBool          fDGPSStatus;
    DWORD           dwDGPSAgeLimit;
    PNAVDGPSSTATUS  dpAvDGPSStatus;
    DWORD           dwPNReserved;
} PNDGPSSTATUS;
```

Members

dwStructureSize
The size, in bytes, of the structure.

tiTime
Time the data was gathered.

DGPSMode

Value	Description
PN_3S_FALSE	DGPS off
PN_3S_TRUE	DGPS on
PN_3S_OTHER	Auto selection

OperatingMode

Value	Description
PN_3S_FALSE	2D only
PN_3S_TRUE	3D only
PN_3S_OTHER	Auto selection

fDGPSSstatus
True, if outputting position with the receiver using DGPS corrections.
False, if not using DGPS corrections.

dwDGPSAgeLimit
Maximum age to use, in milliseconds.

dpAvDGPSStatus

dwPNReserved
Reserved for future use.

PNEnv_t

Pre-defined environments to which P&N devices can be set.

Constant	Value	Meaning
PN_ET_STATIONARY	MIN_PNENV_T	Device is not moving.
PN_ET_OPENROAD	1	Device is on open road with clear view of sky.
PN_ET_URBANCANYON	2	Device is surrounded by tall city buildings. This is the 'City' option in the GPS Control panel applet.
PN_ET_FOREST	3	Device is in a forest – or near trees.
PN_ET_OPENOCEAN	4	Device is on the open ocean with full view of sky. This is the 'Open water' option in the GPS Control panel applet.
PN_ET_AIRCRAFT	5	Device is in an aircraft with full view of sky.
PN_ET_NONE	6	No environment yet set (only returned by PNSETTINGS).
PN_ET_USER	MAX_PNENV_T	TBD.

PNINDSTATION

Stores individual station details and the time each was last updated.

```
typedef struct tagPNINDSTATION
{
    DWORD               dwStructureSize;
    PNTIME              tiTime;
    PNStationState_t    ssState;
    DWORD               dwStationIDNum;
    PNBool              fUsed;
    PNDouble            dSatElevation;
    PNDouble            dSatAzimuth;
    PNDouble            dSignalStrength;
    DWORD               dwCoverage;
    PNAVINDSTATION      siAvIndStation;
    DWORD               dwPNReserved;
} PNINDSTATION;
```

Members dwStructureSize
The size, in bytes, of the structure.
tiTime
Not used by Windows CE.
ssState
State of this station.
dwStationIDNum
PRN#/SVID or unique station number.
fUsed
Whether station is being used for calcns.
dSatElevation
Measured in radians (0-π/2).
dSatAzimuth
Measured in radians (0-2π).
dSignalStrength
Signal strength, in dB.
dwCoverage
Not used by Windows CE.
siAvIndStation
Shows which PNINDSTATION elements are valid and which are not.
dwPNReserved
Reserved for future use by PNAPI.

Remarks For GPS receivers, *dwStationID* is defined as the PRN or SVID satellite number. Numbers 33-64 are reserved for WAAS. Numbers 65-96 are reserved for GLONASS.

If *dwCoverage* is zero, the period of coverage is not available, or is unreliable (i.e. highly variable).

PNPOSITION

Stores the position and time at which this position was found.

```
typedef struct tagPNPOSITION
{
    DWORD       dwStructureSize;
    PNTIME      tiTime;
    PNPOSLLA    psPosition;
    PNAVPOSLLA  psAvPosition;
    DWORD       dwPNReserved;
} PNPOSITION;
```

Members dwStructureSize
The size, in bytes, of the structure.

tiTime
Time the position was acquired.
psPosition
The position.
psAvPosition
Which PNPOSLLA elements are valid.
dwPNReserved
Reserved for future use.

PNPOSLLA

Contains position details in Longitude, Latitude and Altitude units. This is the standard units for the PNAPI.

```
typedef struct tagPNPOSLLA
{
    PNDouble    dLong;
    PNDouble    dLat;
    PNDouble    dAlt;
    PNBool      fRadians;
} PNPOSLLA;
```

Members

dLong
The longitude.
dLat
The latitude.
dAlt
Height above geoid in meters.
fRadians
TRUE if position (dLong and dLat) is in radians, FALSE if in degrees. Position is generally described in radians throughout PNAPI unless otherwise stated.

PNPowerState_t

Enumerates the different power states the P&N device can have.

Constant	Value	Meaning
PN_PW_OFF	MIN_PNPOWERSTATE_T	No power.
PN_PW_SUSPENDED	1	Device temporarily suspended.
PN_PW_STANDBY	2	Device in standby mode.
PN_PW_LOWPOWER	3	Device in low power mode.

PN_PW_MIDPOWER	4	Device in half power mode.
PN_PW_FULLPOWER	5	Device in full power mode.

PNRTCM1

This structure contains the RTCM message.

```
typedef struct PNRTCM1
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    BYTE            ucRTCMMajorVersion;
    BYTE            ucRTCMMinorVersion;
    PNRTCMHEADER    Header;
    BYTE            ucNumSats;
    PNRTCM1SAT      SatData
                    (PN_NUM_RTCM1_SATS_c);
    PNByte          bRawData
                    (PN_RTCM1_MAX_BYTE_LEN_
                    c);
    DWORD           dwPNReserved;
} PNRTCM1;
typedef PNRTCM1* pPNRTCM1;
```

Members

dwStructureSize
: Size of the structure.

tiTime
: The time (as a PNTIME structure).

ucRTCMMajorVersion
: Major version number.

ucRTCMMinorVersion
: Minor version number.

Header
: Message header.

ucNumSats
: Number of valid satellites in SatData.

SatData
: The satellite data.

bRawData
: The raw data.

Remarks

This structure definition is provided for the use of application developers implementing DGPS objects.

PNRTCM1SAT

This structure contains satellite data for DGPS.

```
typedef struct PNRTCM1SAT
{
        DWORD     dwStructureSize;
        PNBool    fScaleFactor;
        BYTE      ucUDRE;
        BYTE      ucSatelliteID;
        WORD      uPsCorrection;
        BYTE      ucRRateCorrection;
        BYTE      ucIssueOfData;
        DWORD     dwPNReserved;
} PNRTCM1SAT;
```

Members

dwStructureSize
Size of the structure.

fScaleFactor

ucUDRE

ucSatelliteID
Satellite ID.

uPsCorrection

ucRRateCorrection

ucIssueOfData

Remarks

This structure definition is provided for the use of application developers implementing DGPS objects.

PNRTCMHEADER

This structure contains the header for an RTCM message.

```
typedef struct tagPNRTCMHEADER
{
        DWORD     dwStructureSize;
        BYTE      ucMessageType;
        WORD      uStationID;
        WORD      uModZCount;
        BYTE      ucSequenceNum;
        BYTE      ucFrameLength;
        BYTE      ucStationHealth;
        DWORD     dwPNReserved;
} PNRTCMHEADER;
```

Members dwStructureSize
Size of the structure.
ucMessageType
Message type (frame ID).
uStationID
Station ID.
uModZCount
??
ucSequenceNum
Sequence number.
ucFrameLength
Frame length.
ucStationHealth
Station health.

Remarks This structure definition is provided for the use of application developers implementing DGPS objects.

PNSATELLITE

Stores individual satellite data.

```
typedef struct tagPNSATELLITE
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    PNBool          fSetData;
    DWORD           dwPRN;
    PNByte          bSatHealth;
    DWORD           dwRefWeekNumber;
    DWORD           dwRefTimeOfWeek;
    PNDouble        dEccentricity;
    PNDouble        dRootSemiMajorAxis;
    PNDouble        dArgumentOfPerigee;
    PNDouble        dMeanAmomalyAtRefTime;
    PNDouble        dRightAscensionAtRefTime;
    PNDouble        dRateRightAscension;
    PNDouble        dCorrectToInclination;
    PNDouble        dAF0ClockCorrect;
    PNDouble        dAF1ClockCorrect;
    PNAVSATELLITE   saAvSatellite;
    DWORD           dwPNReserved;
} PNSATELLITE;
```

Members dwStructSize
The size, in bytes, of the structure.

tiTime
Not used by Windows CE.
fSetData
Not used by Windows CE.
dwPRN
Satellite PRN number.
bSatHealth
Health summary (binary).
dwRefWeekNumber
GPS week number.
dwRefTimeOfWeek
Almanac reference time.
dEccentricity
Eccentricity.
dRootSemiMajorAxis
Measures in meters^0.5.
dArgumentOfPerigee
Measured in radians.
dMeanAnomolyAtRefTime
Measured in radians.
dRightAscensionAtRefTime.
Measured in radians.
dRateRightAscension
Measured in radians/sec.
dCorrectToInclination
Measured in PI radians.
dAF0ClockCorrect
Measured in seconds.
dAF1ClockCorrect
Measured in sec/sec.
saAvSatellite
Which elements are valid.
dwPNReserved
Reserved for future use by PNAPI.

Remark

The *fSetData* parameter is used in the pnapiSetData function. If set, it updates the GPS receiver's almanac with this satellite's data. If not, this structure is not sent to the GPS receiver. When this structure is received through the pnapiGetData or pnapiStartDirectCall function, the *fSetData* parameter has no meaning and should be set to zero.

PNSETTINGS

Stores P&N device settings that can be changed by the user.

typedef struct tagPNSETTINGS

```
{
    DWORD           dwStructureSize;
    PNTIME          tiTime;
    PNSTATIONMODE   cmMode[PN_NUM_SATS_c];
    PNBool          fDGPSEnable;
    PNBool          fDREnable;
    DWORD           dwDGPSTimeOut;
    PNBool          fDGPS2DEnable;
    DWORD           dwDGPS2DTimeOut;
    PNDatum_t       daDatum;
    PNPowerState_t  pwPowerState;
    PNAltHold_t     ahAltitudeHold;
    PNDouble        dAHAltitude;
    PN2DMode_t      mo2DPosMode;
    PNDouble        d2DAltitude;
    PNAccess_t      asAccess; //
    PNEnv_t         etEnvironment;
    PNAVSETTINGS    seAvSettings; //
    DWORD           dwPNReserved; //
} PNSETTINGS;
```

Members

dwStructureSize
The size, in bytes, of the structure.

tiTime
The time when the data was gathered.

cmMode
Not used by Windows CE.

fDGPSEnable
Enables/disables DGPS functionality.

fDREnable
Enable/disables dead reckoning functionality.

dwDGPSTimeOut
Sets/gets the DGPS time out (in milliseconds).

fDGPS2DEnable
Not used by Windows CE.

dwDGPS2DTimeOut
Not used by Windows CE.

daDatum
Datum receiver uses.

pwPowerState
Power state of device.

ahAltitudeHold
Not used by Windows CE.

dAHAltitude
Not used by Windows CE.

mo2DPosMode
Not used by Windows CE.

d2DPosMode
Not used by Windows CE.
d2DAltitude
Not used by Windows CE.
asAccess
Access rights for device.
etEnvironment
Environment for this device.
seAvSettings
Which elements are valid.
dwPNReserved
Reserved for future use by PNAPI.

PNSTATION

Contains the details for all stations the P&N device has access to.

```
typedef struct tagPNSTATION
{
    DWORD       dwStructureSize;
    PNTIME      tiTime;
    DWORD       dwNumAvailable; //
    DWORD       dwNumUsed;
    PNAVSTATION snAvStation;
    PNINDSTATION siStations
                [PN_NUM_STATIONS_c];
    DWORD       dwPNReserved;
} PNSTATION;
```

Members

dwStructureSize
The size, in bytes, of the structure.
tiTime
The time the structure was last updated.
dwNumAvailable
Not used by Windows CE.
dwNumUsed
Number of stations being tracked by the device.
snAvStation
Stores which elements of PNSTATION of are valid and which are not.
siStations
Individual station data.
dwPNReserved
Reserved for future use by PNAPI.

PNStationState_t

Enumerates the station states.

Constant	Value	Description
PN_CS_UNAVAILABLE	0	Station unavailable.
PN_CS_IDLE	1	Station idle.
PN_CS_SEARCHING	2	Station searching for data.
PN_CS_TRACKING	3	Station finding good data.

PNTIME

Stores P&N device time and computer system time.

```
typedef struct tagPNTIME
{
    PNTM      tmDevice;
    PNAVTM    tmAvDevice;
    PNTM      tmLeapDiffTime;
    PNAVTM    tmAvLeapDiffTime;
    PNTM      tmComputer;
    PNAVTM    tmAvComputer;
} PNTIME;
```

Members

tmDevice
The time reported by the device.

tmAvDevice
Stores which elements of tmAvDevice are valid and which are not.

tmLeapDiffTime
Not used by Windows CE.

tmAvLeapDiffTime
Not used by Windows CE.

tmComputer
The system time on the computer.

tmAvComputer
Stores which elements of tmAvComputer are valid and which are not.

PNTM

Stores time to the millisecond.

```
typedef struct tagPNTM
{
        DWORD           dwMillisec;
        DWORD           dwDay;
} PNTM;
```

Members

dwMillisec
Milliseconds since start of day (0-86400000).

dwDay
Days since Jan 1st 1900.

PNVELBEAR

Contains velocity details in the form of a bearing and two velocities.

```
typedef struct tagPNVELBEAR
{
        PNDoubledBearing;
        PNDoubledHorizSpeed;
        PNDoubledVertSpeed;
} PNVELBEAR;
```

Members

dBearing
dBearing has a range from –PI to +PI. Zero is North.

dHorizSpeed
Horizontal speed in meters/sec.

dVertSpeed
Vertical speed in meters per second.

PNVELENU

Contains velocity details in the East, North, Up format.

```
typedef struct tagPNVELENU
{
        PNDouble    East;
        PNDouble    North;
        PNDouble    Up;
} PNVELENU;
```

Members

East
East velocity, in meters/second.

North
North velocity, in meters/second.

Up

Up velocity, in meters/second.

Remarks A westward velocity is expressed as a negative East velocity and a southward velocity is expressed as a negative North velocity.

PNVELOCITY

Stores velocities and the time they were last updated.

```
typedef struct tagPNVELOCITY
{
    DWORD       dwStructureSize;
    PNTIME      tiTime;
    PNVELENU    vlVelocity;
    PNAVVELENU  vlAvVelocity;
    DWORD       dwPNReserved;
} PNVELOCITY;
```

Members

dwStructureSize
The size, in bytes, of the structure.

tiTime
The time.

vlVelocity
The velocity.

vlAvVelocity
Shows which vlVelocity elements are valid and which are not.

dwPNReserved
For future use.

Detailed Description of a Handwriting Recognition API

Module/component:
Platforms: H/PC
Windows CE versions: 2.02 and later

Parameters *hVol*
VOL structure returned from FSDMGR_RegisterVolume.
hProc
Originating process handle.
pSearch
FSD-defined search-specific data for the new handle.

Return Values If the function is successful, it returns a search handle associated with the originating process. If it is unsuccessful, it returns INVALID_HANDLE_VALUE.

Remarks FSDMGR_RegisterVolume

See Also

HwxConfig

The HwxConfig function initializes the handwriting recognition dynamic-link library (DLL).

Syntax
```
BOOL HwxConfig (
void
);
```

At a Glance Header file: Recog.h
Module/component:
Platforms: H/PC
Windows CE versions: 2.0 and later

Return Values If the function is successful, it returns TRUE. If an error occurred initializing the handwriting recognition engine, the function returns FALSE.

If it is unsuccessful, use GetLastError to identify the cause of the error.

Remarks This function is called only once by each application to initialize the DLL.

HwxCreate

The HwxCreate function creates a handwriting recognition context (HRC) object for the recognizer.

Syntax

```
HRC HwxCreate (
        HRC hrc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hrc

Handle to an existing HRC object that provides settings for the recognition context being created. If it is NULL, then default settings are used.

Return Values

If the function is successful, it returns the handle to the newly created HRC object; otherwise, it returns NULL.

If HwxCreate fails, use GetLastError to get error information.

Remarks

This function is called before any ink is collected.

The *hrc* parameter is used to copy an old context's settings into the new HRC object. These settings include word lists, coercion, and the HWXGUIDE structure, but exclude any pen data that may be in the old context.

See Also

HwxDestroy, HWXGUIDE

HwxDestroy

The HwxDestroy function destroys a handwriting recognition context (HRC) object.

Syntax

```
BOOL HwxDestroy (
        HRC hrc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hrc*
Handle to the HRC object.

Return Values If the function is successful, it returns TRUE. If there was an invalid parameter or other error, it returns FALSE.

If this function fails, call GetLastError for error information.

Remarks This function is called to destroy an HRC after recognition is complete. After HwxDestroy returns TRUE, the handle *hrc* is no longer valid. The application should set *hrc* to NULL to ensure it is not inadvertently used again.

HwxSetGuide

The HwxSetGuide function identifies the location of the boxes on the screen for a specified handwriting recognition context (HRC).

Syntax
```
BOOL HwxSetGuide (
    HRC hrc,
    HWXGUIDE* lpGuide
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hrc*
Handle to the HRC object.
lpGuide
Pointer to a HWXGUIDE structure.

Return Values If the function is successful, it returns TRUE. If the function is unsuccessful, it returns FALSE.

If the function fails, use GetLastError to get error information.

Remarks This function is used for doing boxed recognition. The GUIDE structure defines the size and position of the boxes. If *lpGuide* is NULL, or if all the members in the GUIDE structure are 0, the recognizer does not use guides. This is also known as free input.

See Also HWXGUIDE

HwxALCValid

The HwxALCValid function defines the set of characters that the recognizer can return.

Syntax

```
BOOL HwxALCValid (
    HRC hrc,
    ALC alc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	P/PC
Windows CE versions:	2.0 and later

Parameters

hrc

Handle to the handwriting recognition context (HRC) object.

alc

ALC value that describes the character grouping that is used by the recognizer to evaluate the input handwriting. It can be one or more of the following values:

ALC_WHITE
White space.

ALC_LCALPHA
The lowercase alphabet, a through z.

ALC_UCALPHA
The uppercase alphabet, A through Z.

ALC_NUMERIC
0 through 9.

ALC_PUNC
Standard punctuation, language dependent.

ALC_NUMERIC_PUNC
Non-digit characters in numbers.

ALC_MATH
%^*()_+{}</ (???Language dependent???)

ALC_MONETARY
Punctuation in local monetary expressions.

ALC_COMMON_SYMBOLS
Commonly used symbols from all categories.

ALC_OTHER
Other punctuation not typically used.

ALC_ASCII
7-bit characters – 20 through 7F.

ALC_HIRAGANA
Hiragawa.

ALC_KATAKANA
Katakana.

ALC_KANJI_COMMON
Common Kanji (JPN).
ALC_KANJI_RARE
ALC_HANGUL_COMMON
Common Hangul used in Korea.
ALC_HANGUL_RARE
The rest of Hangul used in Korea.
ALC_UNUSED
Reserved for future use.
ALC_OEM
OEM recognizer specific.

Useful groupings, by combining two or more of the basic ALC groupingsuseful ALC groupings	definition
ALC_ALPHA	ALC_LCALPHA | ALC_UCALPHA
ALC_ALPHANUMERIC	ALC_ALPHA | ALC_NUMERIC
ALC_KANA	ALC_HIRAGANA | ALC_KATAKANA
ALC_KANJI_ALL	ALC_KANJI_COMMON | ALC_KANJI_RARE
ALC_HANGUL_ALL	ALC_HANGUL_COMMON | ALC_HANGUL_RARE
ALC_EXTENDED_SYM	ALC_MATH | ALC_MONETARY | ALC_OTHER
ALC_SYS_MINIMUM	ALC-ALPHANUMERIC | ALC_PUNC | ALC_WHITE
ALC-SYS-DEFAULT	ALC_SYS_MINIMUM | ALC_COMMON_SYMBOLS

Standard combinations for various languages.language ALC groupings	definition
ALC_USA_COMMON	ALC_SYS_DEFAULT
ALC_USA_EXTENDED	ALC_USA_COMMON | ALC_EXTENDED_SYM
ALC_JPN_COMMON	ALC_SYS_DEFAULT | ALC_KANA | ALC_KANJI_COMMON
ALC_JPN_EXTENDED	ALC_JPN_COMMON | ALC_EXTENDED_SYM | ALC_KANJI_RARE
ALC_CHS_COMMON	ALC_SYS_DEFAULT | ALC_KANJI_COMMON
ALC_CHS_EXTENDED	ALC_CHS_COMMON | ALC_EXTENDED_SYM | ALC_KANJI_RARE
ALC_CHT_COMMON	ALC_SYS_DEFAULT | ALC_KANJI_COMMON

ALC_CHT_EXTENDED	ALC_CHT_COMMON | ALC_EXTENDED_SYM | ALC_KANJI_RARE
ALC_KOR_COMMON	ALC_SYS_DEFAULT | ALC_HANGUL_COMMON | ALC_KANJI_COMMON
ALC_KOR_EXTENDED	ALC_KOR_COMMON | ALC_EXTENDED_SYM | ALC_HANGUL_RARE | ALC_KANJI_RARE

Return Values If the recognizer is set to recognize the specified ALC grouping, the function returns TRUE. If the recognizer is not set, the function returns FALSE.

If HwxALCValid fails, use GetLastError for error information.

Remarks This function tells the recognizer which characters to use to evaluate the ink in the HRC.

HwxALCPriority

The HwxALCPriority function reorders the characters returned by the recognizer so that selected characters appear at the top of the list.

Syntax

```
BOOL HwxALCPriority (
    HRC hrc,
    ALC alc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hrc

Handle to the handwriting recognition context (HRC) object.

alc

ALC value that describes the character grouping that will be used by the recognizer to ???????.

Return Values If the recognizer has been reset for the selected characters, the function returns TRUE. The function returns FALSE otherwise.

If this function fails, use GetLastError to identify the cause of the error.

Remarks ????????? need to describe how this works ??????????

See Also HwxALCValid

HwxSetPartial

The HwxSetPartial function sets the recognizer parameter for partial recognition.

Syntax
```
BOOL HwxSetPartial (
        HRC hrc,
        UINT urecog
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hrc

Handle for the recognition context (HRC) object.

urecog

Value for the partial recognition parameter. It can be one of the following values:

?????????????

Return Values If the recognizer is set with the partial recognition value, the function returns TRUE. The function returns FALSE otherwise.

If HwxSetPartial fails, use GetLastError for error information.

Remarks ?????????? describe partial recognition ????????????

HwxSetAbort

The HwxSetAbort function sets the abort address.

Syntax
```
BOOL HwxSetAbort (
        HRC hrc,
        void** ppabortaddr
);
```

At a Glance
Header file: Recog.h
Module/component:
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
hrc
Handle of the handwriting recognition context (HRC) object.
ppabortaddr
???????? pointer to a pointer to the abort address ????????

Return Values If the recognizer is set with the abort address, the function returns TRUE. The function returns FALSE otherwise.

If HwxSetAbort fails, use GetLastError for error information.

Remarks ????????? describe why you use this ????????????

HwxInput

The HwxInput function adds ink to the handwriting recognition context (HRC).

Syntax

```
BOOL HwxInput (
    HRC hrc,
    POINT* lppnt,
    UINT upoints,
    DWORD timestamp
);
```

At a Glance
Header file: Recog.h
Module/component:
Platforms: H/PC
Windows CE versions: 2.0 and later

Parameters
hrc
Handle to the HRC object.
lppnt
Address of an array of POINT structures. The information in the POINT structures should be scaled to match the HWXGUIDE structure.
upoints
Number of POINT structures.
timestamp
Time stamp of the first mouse event in the stroke. The time stamp should be taken directly from the MSG structure for the mouse down event.

Return Values If the function is successful, it returns TRUE. If there is an invalid parameter or other error, it returns FALSE.

If this function fails use GetLastError for error information.

Remarks This function adds ink to the HRC object one stroke at a time. It takes the array of points, the count of the points, and the time stamp of the first mouse event in the stroke and adds it to the HRC object.

See Also HWXGUIDE, POINT

HwxEndInput

The HxwEndInput function tells the recognizer that no more ink should be added to the handwriting recognition context (HRC) object.

Syntax
```
BOOL HwxEndInput (
        HRC hrc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hrc*

Handle to the HRC object that is to be closed.

Return Values If the HRC is closed, the function returns TRUE; otherwise, it returns FALSE.

Remarks This function is called after the last ink is added to the HRC. The next call to HwxProcess completes recognition on all the input. Any calls to HwxInput on this HRC fail after HwxEndInput is called.

See Also HwxInput, HwxProcess

HwxProcess

The HwxProcess function signals the recognizer to analyze the information in the specified handwriting recognition context (HRC) object.

Syntax

```
BOOL HwxProcess (
        HRC hrc
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hrc

Handle to the HRC object to be analyzed.

Return Values If the recognition is completed, the function returns TRUE. If there is an invalid parameter or other error, it returns FALSE.

Remarks This function processes the ink that has been received by the HRC object. Full recognition occurs only after HwxEndInput is called. The application must then call HwxGetResults to obtain recognition results.

There is no support for timeouts.

If the function fails, use GetLastError for error information.

See Also HwxEndInput, HwxGetResults

HwxGetResults

The HwxGetResults function retrieves the results from the recognition on the handwriting recognition context (HRC).

Syntax

```
INT32 HwxGetResults (
        HRC hrc,
        UINT cAlt,
        UINT iFirst,
        UINT cBoxRes,
        HWXRESULTS *rgBoxResults
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters

hrc
Handle to the HRC object used for input.

cAlt
Number of alternate results expected in the HWXRESULTS structure. If this parameter is 0, the function returns 0.

iFirst
Index of the first character to return.

cBoxRes
Number of characters to return.

rgBoxResults
Array of *cBoxRes*-ranked lists.

Return Values If the function is successful, it returns the number of characters actually returned; otherwise, it returns HRCR_ERROR, which indicates an invalid parameter or other error.

Remarks This function retrieves the results from an HRC object used for boxed input. It simplifies the task of boxed recognition by providing character alternatives on a per-box basis in one call. This function may be called repeatedly, allowing you to get results for several characters at a time. The results for the returned characters are put in the *rgBoxResults* buffer that was passed in.

See Also HWXRESULTS

HwxSetContext

The HwxSetContext function adds context information to the handwriting recognition context (HRC).

Syntax

```
BOOL HwxSetContext (
        HRC hrc,
        WCHAR WchContext
);
```

At a Glance

Header file:	Recog.h
Module/component:	
Platforms:	H/PC
Windows CE versions:	2.0 and later

Parameters *hrc*

Handle to the HRC object.

WchContext

Character of prior context to the characters contained in the HRC. If this parameter is 0, it clears the context information.

Return Values This function returns TRUE if successful; if there was an invalid parameter or other error, it returns FALSE.

If the function fails, use GetLastError for error information.

Remarks Handwriting recognition performance can be improved if the recognizer has context information available during processing. Context information is added to an HRC by using HwxSetContext, which provides one character of prior context for the recognizer. This function should be called prior to using the HwxProcess function. If this function is not called, the recognizer assumes that no prior context is available.

See Also HwxProcess

HwxResultsAvailable

The HwxResultsAvailable function returns the number of characters available for HwxGetResults to retrieve.

Syntax

```
INT HwxResultsAvailable (
    HRC hrc
);
```

At a Glance		
	Header file:	Recog.h
	Module/component:	
	Platforms:	H/PC
	Windows CE versions:	2.0 and later

Parameters *hrc*

Handle to the handwriting recognition context (HRC) object.

Return Values Number of characters available for HwxGetResults to retrieve. It returns –1 on error.

If the function fails, use GetLastError for error information.

Remarks This function allows characters to be retrieved before all the input has been added to the HRC.

See Also HwxGetResults

GetThreadTimes

The GetThreadTimes function obtains timing information about a specified thread.

Syntax

```
BOOL GetThreadTimes (
    HANDLE hThread
    LPFILETIME lpCreationTime,
    LPFILETIME lpExitTime,
    LPFILETIME lpKernelTime,
    LPFILETIME lpUserTime
);
```

At a Glance	
Header file:	Winbase.h
Module/component:	
Platforms:	H/PC

Detailed Description of a Speech-to-Text API

CHAPTER 5

IVoiceText

The IVoiceText interface registers an application to use the voice-text object, and controls playback of text.

Method	Description
IVoiceText::AudioFastForward	Unsupported
IVoiceText::AudioPause	Pauses text-to-speech output
IVoiceText::AudioResume	Resumes text-to-speech output
IVoiceText::AudioRewind	Unsupported
IVoiceText::Register	Registers an application to use voice text
IVoiceText::Speak	Starts playing the specified text
IVoiceText::StopSpeaking	Halts text that is currently being spoken

IVoiceText::AudioPause

Pauses text-to-speech output for a voice-text site.

Syntax HRESULT AudioPause(void);

Parameters None

Return Values This method returns NOERROR if successful, or one of these error values:

VTXTERR_INVALIDMODE
VTXTERR_NOTENABLED
VTXTERR_OUTOFMEM

Remarks AudioPause affects all applications using the site, so the application should resume audio as soon as possible.

When a voice-text object is first created, text-to-speech output is not paused. Because pausing text-to-speech output affects all applications that use voice text on the site, an application should resume text-to-speech output as soon as possible by calling the IVoiceText::AudioResume member function.

When output has been paused, the IVTxtAttributes::IsSpeaking member function returns FALSE, even though the voice-text object still has data available in its queue and has not yet sent a IVTxtNotifySink::SpeakingDone notification.

No notifications are sent when audio is paused or resumed.

See Also IVoiceText::AudioResume, IVTxtAttributes::IsSpeaking, IVTxtNotifySink::SpeakingDone

IVoiceText::AudioResume

Resumes text-to-speech output after it has been paused by the IVoiceText::AudioPause member function.

Syntax HRESULT AudioResume(void);

Parameters None

Return Values This method returns NOERROR if successful, or one of these error values:

VTXTERR_INVALIDMODE
VTXTERR_NOTENABLED
VTXTERR_OUTOFMEM

Remarks AudioResume affects all applications using the site.

See Also IVoiceText::AudioPause

IVoiceText::Register

Registers an application to use voice text on a site.

Syntax
```
HRESULT Register (
    PTSTR pszSite,
    PTSTR pszApplication,
    PIVTXTNOTIFYSINK pNotifyInterface,
    IID IIDNotifyInterface,
    DWORD dwFlags,
    PVTSITEINFO pSiteInfo
);
```

Parameters *pszSite*

For Auto PC, must be null or empty.

pszApplication

[in] Address of a string that identifies the application – for example, "Microsoft Word." An application can use this information to display the source of text. This parameter must not be NULL.

pNotifyInterface

[in] Address of the notification interface through which the voice-text object notifies the application about text-to-speech information. If this parameter is NULL, no notifications will be sent. The interface identifier is specified by *IIDNotifyInterface*.

Because passing the pointer to the voice-text object does not transfer ownership of the notification interface, the voice-text object must call the AddRef member function of the notification interface before returning from the call to Register. The voice-text object must also call the Release member function of the notification interface when it closes. The calling application must release any reference counts it holds on the notification interface after calling Register, unless it needs the notification object to be valid when the voice-text object releases it.

IIDNotifyInterface

[in] GUID of the interface used for notification. For Auto PC, this parameter must be IID_IVTxtNotifySinkW (for Unicode).

dwFlags

[in] Flag that indicates whether the application is to receive all notifications. If this parameter is the VTXTF_ALLMESSAGES value, all notifications are sent to *pNotifyInterface*. If this parameter is zero (0) or null, only the IVTxtNotifySink::SpeakingStarted and IVTxtNotifySink::SpeakingDone notifications are sent.

pSiteInfo

[in] Address of a VTSITEINFO structure that contains settings to apply to the site, such as the voice and talking speed. The settings are applied, even if the site is already open. If a VTSITEINFO structure is not specified, the voice-text object uses the settings from the registry. If there are no registry settings, it uses the default settings, typically those for the computer.

Telephony applications pass this information to ensure that the proper settings are selected. Other applications will set this parameter to NULL to leave the site settings unchanged.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks An application must call Register before it can call other functions in the IVoiceText interface.

An application cannot call Register a second time for the same voice-text object. To change sites, the application must call the CoCreateInstance function to create a new voice-text object for the desired site.

See Also VTSITEINFO, IVTxtNotifySink::SpeakingStarted, IVTxtNotifySink:: Speaking Done

IVoiceText::Speak

Starts playing the specified text.

Syntax HRESULT Speak(
PTSTR *pszSpeak*,
DWORD *dwFlags*,
PTSTR *pszTags*
);

Parameters *pszSpeak*
[in] Address of a buffer that contains the text to speak. An application can free or modify the buffer as soon as Speak returns. The string pointed to by this parameter can contain text-to-speech control tags.

dwFlags
[in] Flags that indicate the type and priority of the text. This parameter is a combination of one type flag and one priority flag.

The type flag can be one of these values:

VTXTSP_HIGH
Play the text as soon as possible, after text that is currently being spoken but before any other text in the playback queue.

VTXTSP_NORMAL
Play the text immediately, interrupting text that is currently being spoken, if any. The interrupted text resumes playing as soon as the very high priority text is finished, although the interrupted text may not be correctly synchronized.

pszTags
[in] Address of a buffer that contains text-to-speech control tags to change the voice, language, or context of the text specified by *pszSpeak*, or NULL to use the default settings for the text-to-speech voice. For more

information about control tags, see Appendix A, "Text-to-Speech Control Tags."

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_NOTENABLED
- VTXTERR_OUTOFMEM
- VTXTERR_QUEUEFULL
- VTXTERR_WAVEDEVICEBUSY

Remarks If an application calls Speak when other text is being played, the specified text is added to the end of the playback queue, unless the application specifies a higher priority in *dwFlags*.

Calling Speak affects all applications using voice text on the site, because all applications share the same playback queue.

The type of speech specified by *dwFlags* is communicated to the text-to-speech engine through control tags. Support of most control tags is optional; the engine ignores unsupported tags.

See Also IVoiceText::StopSpeaking

IVoiceText::StopSpeaking

Halts text that is currently being spoken and flushes all pending text from the playback queue.

Syntax HRESULT StopSpeaking(void);

Parameters None

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_NOTENABLED
- VTXTERR_OUTOFMEM

Remarks Calling StopSpeaking affects all applications using voice text on the site, because all applications share the same playback queue.

See Also IVoiceText::Speak

IVTxtAttributes

The IVTxtAttributes interface allows an application to control various aspects of the operation of a Voice Text object.

Method	Description
IVTxtAttributes::DeviceGet	Not Implemented
IVTxtAttributes::DeviceSet	Not Implemented
IVTxtAttributes::EnabledGet	Discovers whether voice text is enabled.
IVTxtAttributes::EnabledSet	Enables or disables voice text.
IVTxtAttributes::IsSpeaking	Indicates whether text is currently being spoken.
IVTxtAttributes::SpeedGet	Retrieves the current average talking speed.
IVTxtAttributes::SpeedSet	Sets the average talking speed.
IVTxtAttributes::TTSModeGet	Retrieves the current text-to-speech mode.
IVTxtAttributes::TTSModeSet	Sets the text-to-speech mode.

IVTxtAttributes::EnabledGet

Discovers whether voice text is enabled for a voice-text site.

Syntax

```
HRESULT EnabledGet(
        DWORD *dwEnabled
);
```

Parameters

dwEnabled

[out] TRUE if voice text is enabled for the site or FALSE if it is disabled.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks

If voice text is disabled, no text-to-speech is played over the site. Enabling or disabling voice text for a site affects all applications using a voice-text site.

Typically, an application disables voice text because the user does not want the computer to speak. You should involve the user when enabling or disabling voice text.

The enabled state for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

See Also IVTxtAttributes::EnabledSet

IVTxtAttributes::EnabledSet

Enables or disables voice text for a voice-text site.

Syntax
```
HRESULT EnabledSet(
    DWORD dwEnabled
);
```

Parameters *dwEnabled*
[in] TRUE to enable voice text or FALSE to disable it.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks The enabled state for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

If a voice-navigation application is installed on the user's computer, an application may not need to set the enabled state.

See Also IVTxtAttributes::EnabledGet

IVTxtAttributes::IsSpeaking

Indicates whether text is currently being spoken by a voice-text site.

Syntax
```
HRESULT IsSpeaking(
    BOOL *pfSpeaking
);
```

Parameters *pfSpeaking*
[out] Address of a variable that receives the current speaking status. The variable receives TRUE if the text-to-speech engine is speaking or FALSE if it is silent.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks — The voice text object does not send data resulting from multiple calls to the IVoiceText::Speak member function directly to the text-to-speech engine. Instead, the object keeps data from each call in a separate buffer so that the VTXTSP_HIGH and VTXTSP_VERYHIGH priority strings can be inserted into the queue at the proper positions.

For example, a VTXTSP_VERYHIGH priority string may interrupt a high or normal priority string. The interrupted string resumes after the very high priority string has finished. As a result of this implementation, IsSpeaking returns FALSE for a short time between the end of one buffer in the queue and the start of the next buffer, because audio output has been temporarily suspended.

IVTxtAttributes::SpeedGet

Retrieves the current average talking speed for a voice-text site, in words per minute.

Syntax

```
HRESULT SpeedGet(
        DWORD *pdwSpeed
);
```

Parameters

pdwSpeed

[out] Address of a variable that receives the talking speed for a voice-text site.

Return Values — This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks — The talking speed for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

See Also — IVTxtAttributes::SpeedSet

IVTxtAttributes::SpeedSet

Sets the average talking speed for a voice-text site, in words per minute.

Syntax

```
HRESULT SpeedSet(
    DWORD dwSpeed
);
```

Parameters

dwSpeed

[in] New talking speed for the site. An application can specify TTSATTR_MINSPEED or TTSATTR_MAXSPEED for the minimum or maximum allowable value.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks

The talking speed for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

If a voice-navigation application is installed on the user's computer, an application may not need to set the speed.

See Also IVTxtAttributes::SpeedGet

IVTxtAttributes::TTSModeGet

Retrieves the GUID of the current text-to-speech mode for a voice-text site.

Syntax

```
HRESULT TTSModeGet(
    GUID *pgVoice
);
```

Parameters

pgVoice

[out] Address of a variable that receives the GUID assigned to the text-to-speech mode.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks A text-to-speech engine typically provides an assortment of text-to-speech modes that can be used to play speech in different voices. A voice-text site uses a single text-to-speech mode, represented internally by a low-level engine object.

The text-to-speech mode for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

In Auto PC, there is usually only one TTS mode.

See Also IVTxtAttributes::TTSModeSet

IVTxtAttributes::TTSModeSet

Sets the text-to-speech mode for a voice-text site.

Syntax
```
HRESULT TTSModeSet(
    GUID gVoice
);
```

Parameters *gVoice*

[in] GUID of the text-to-speech mode to set for the site. If the mode does not exist, an error is returned and the mode is not changed.

Return Values This method returns NOERROR if successful, or one of these error values:

- VTXTERR_INVALIDMODE
- VTXTERR_INVALIDPARAM
- VTXTERR_OUTOFMEM

Remarks The text-to-speech mode for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

If a voice-navigation application is installed on the user's computer, an application may not need to set the mode.

In Auto PC, there is usually only one TTS mode.

See Also IVTxtAttributes::TTSModeGet

IVTxtNotifySink

The IVTxtNotifySink interface is used by a Voice Text object to notify an application of the status of the object.

Method	Description
IVTextNotifySink::AttribChanged	Not implemented
IVTextNotifySink::Speak	Used internally
IVTxtNotifySink::SpeakingDone	Speaking is finished
IVTxtNotifySink::SpeakingStarted	Speaking has started
IVTextNotifySink::Visual	Not Implemented

IVTxtNotifySink::SpeakingDone

Notifies all applications on a voice-text site that speaking is finished and no text remains in the playback queue.

Syntax HRESULT SpeakingDone (void);

Parameters None

Return Values The return value is ignored.

See Also IVTxtNotifySink::SpeakingStarted

IVTxtNotifySink::SpeakingStarted

Notifies all applications on a voice-text site that speaking has started.

Syntax HRESULT SpeakingStarted(void);

Parameters None

Return Values The return value is ignored.

See Also IVTxtNotifySink::SpeakingDone

Detailed Description of a Voice Command API

Chapter 4

IVCmdAttributes

The IVCmdAttributes interface provides methods to set various attributes of the Voice Command object, including audio output, recognition mode, and whether or not recognition is enabled.

Method	Description
IVCmdAttributes::AutoGainEnable Get	Not Implemented
IVCmdAttributes::AutoGainEnable Set	Not Implemented
IVCmdAttributes::AwakeStateGet	Retrieves the awake state of a voice-command site.
IVCmdAttributes::AwakeStateSet	Sets the awake state for a voice-command site.
IVCmdAttributes::DeviceGet	Not Implemented
IVCmdAttributes::DeviceSet	Not Implemented
IVCmdAttributes::EnabledGet	Finds out whether speech recognition is enabled or disabled for a voice-command site.
IVCmdAttributes::EnabledSet	Enables or disables speech recognition for a voice-command site.
IVCmdAttributes::MicrophoneGet	Not Implemented
IVCmdAttributes::MicrophoneSet	Not Implemented
IVCmdAttributes::SpeakerGet	Retrieves the name of the current speaker for a voice-command site.
IVCmdAttributes::SpeakerSet	Sets the name of the current speaker for a voice-command site.
IVCmdAttributes::SRModeGet	Retrieves the GUID of the speech-recognition mode used for the site.
IVCmdAttributes::SRModeSet	Sets the speech-recognition mode used by a voice-command site.
IVCmdAttributes::ThresholdGet	Retrieves the threshold level of the speech-recognition engine used by a voice-command site.

Method	Description
IVCmdAttributes::ThresholdSet	Sets the threshold level for the speech-recognition engine used by a voice-command site.

Remarks This interface is supported by all voice-command objects.

IVCmdAttributes::AwakeStateGet

IVCmdAttributes::AwakeStateGet retrieves the awake state for a voice-command site.

Syntax

```
HRESULT AwakeStateGet(
    DWORD *pdwAwake
);
```

Parameters *pdwAwake*

[out] Address of a variable that receives the current state of speech recognition for the site. This parameter is TRUE if the site is awake or FALSE if it is asleep.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks When the site is awake, it listens for commands from any active voice menu for the active application. When the site is asleep, it listens for commands only from sleep menus – those that were activated with the *dwFlags* parameter of the IVCmdMenu::Activate member function set to the VWGFLAG_ASLEEP value. Commands from such menus become active only when the site is asleep, and they become inactive when the site is awake. A sleep menu typically contains a "Wake up!" command that resumes speech recognition, and it may contain other commands.

See Also IVCmdAttributes::AwakeStateSet

IVCmdAttributes::AwakeStateSet

IVCmdAttributes::AwakeStateSet sets the awake state for a voice-command site.

Syntax

```
HRESULT AwakeStateSet(
    DWORD dwAwake
);
```

Parameters

dwAwake

[in] Set to TRUE to cause the site to wake up or FALSE to cause it to go to sleep.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks

If a voice-navigation application is installed on the user's computer, suspending speech recognition by using AwakeStateSet will typically cause the voice-navigation application to activate a "wake up" menu.

Calling AwakeStateSet allows the user to temporarily suspend speech recognition for a site. For example, the user might want to suspend speech recognition from the computer microphone during a telephone conversation and resume recognition when the conversation is finished. The user resumes recognition by speaking an appropriate command from a sleep menu – for example, "Wake up!"

The sleep state for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

If a voice-navigation application is installed on the user's computer, an application may not need to set the sleep state. However, it may call this function to make sure that speech recognition is awake. For example, if an application speaks (with voice text or text-to-speech) "Do you want to print the document?" it might enable and wake up speech recognition for the site to receive the user's reply. The application should then restore speech recognition to its previous state.

IVCmdAttributes::EnabledGet

IVCmdAttributes::EnabledGet finds out whether speech recognition is enabled or disabled for a voice-command site.

Syntax

```
HRESULT EnabledGet(
    DWORD *dwEnabled
);
```

Parameters

dwEnabled

[out] Set to TRUE if speech recognition is enabled for the site or FALSE if it is disabled.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks

When speech recognition is disabled, the engine does not recognize any command from any menu, whether speech recognition is awake or asleep or any menus are active. An application would use the IVCmdAttributes::EnabledSet member function to allow the user to turn speech recognition completely off, as opposed to suspending speech recognition temporarily by putting the site to sleep.

The enabled state for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

IVCmdAttributes::EnabledSet

IVCmdAttributes::EnabledSet enables or disables speech recognition for a voice-command site.

Syntax

```
HRESULT EnabledSet(
    DWORD dwEnabled
);
```

Parameters

dwEnabled

[in] Set to TRUE to enable speech recognition or FALSE to disable it.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG

- VCMDERR_INVALIDMODE
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks

Whenever speech is turned on or off, the WM_SPEECHSTARTED or WM_SPEECHENDED message is sent to all top-level windows in the system. An application can use these messages to determine when to enable or disable its voice commands or voice-text capabilities.

Calling EnabledSet allows the user to completely turn off speech recognition for a site so that nothing is recognized, including commands on sleep menus. For example, the user might want to disable speech recognition from the computer microphone during a meeting so that speech recognition will stay off, even if somebody inadvertently speaks a command on a sleep menu.

If a voice-navigation application is installed on the user's computer, an application may not need to set the enabled state. However, it may call this function to make sure that speech recognition is awake. For example, if an application speaks (with voice text or text-to-speech) "Do you want to print the document?" it might enable and wake up speech recognition for the site to receive the user's reply. The application should then restore speech recognition to its previous state.

Note, however that, if speech recognition is disabled, it is probably because the user does not want to use it. It may not be appropriate to enable speech recognition under those circumstances.

The enabled state for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

IVCmdAttributes::SpeakerGet

IVCmdAttributes::SpeakerGet retrieves the name of the current speaker for a voice-command site.

Syntax

```
HRESULT SpeakerGet(
        PTSTR pszSpeaker,
        DWORD dwSize,
        DWORD *pdwNeeded
);
```

Parameters

pszSpeaker

[in/out] Address of a buffer that receives the name of the current speaker.

dwSize

[in] Size, in bytes, of the buffer specified by *pszSpeaker*. If the buffer is too small, the function returns an error and fills *pdwNeeded* with the number of bytes needed to store the speaker string.

pdwNeeded

[out] Address of a variable that receives the number of bytes needed for the speaker string.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks Changing the speaker name unloads all training for the previous speaker and loads the training for the new speaker. If no training exists for the new speaker, the application starts with default training.

The speaker name for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

IVCmdAttributes::SpeakerSet

IVCmdAttributes::SpeakerSet sets the name of the current speaker for a voice-command site.

Syntax

```
HRESULT SpeakerSet(
        PTSTR pszSpeaker
);
```

Parameters *pszSpeaker*

[in] Address of the string that contains the name of the speaker to set. If the speaker is unknown, this parameter can be an empty string.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks The speaker name for a site is saved between uses of the site, even if the user shuts down the computer in the meantime. The string is not case sensitive.

If a voice-navigation application is installed on the user's computer, an application may not need to set the speaker name.

IVCmdAttributes::SRModeGet

IVCmdAttributes::SRModeGet retrieves the GUID of the speech-recognition mode used for the site.

Syntax
```
HRESULT SRModeGet(
        GUID *pgMode
);
```

Parameters *pgMode*
[out] Address of a variable that receives the unique GUID assigned to the speech-recognition mode.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM

Remarks A speech-recognition engine typically provides an assortment of modes that it can use to recognize speech in different languages or dialects. A voice-command site uses a single speech-recognition mode.

The speech-recognition mode for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

In Auto PC, there is usually only one speech recognition mode.

IVCmdAttributes::SRModeSet

IVCmdAttributes::SRModeSet sets the speech-recognition mode used by a voice-command site.

Syntax
```
HRESULT SRModeSet(
        GUID gMode
);
```

Parameters *gMode*

[in] GUID of the speech-recognition mode to set for the site. If the mode does not exist, an error is returned and the mode is not changed.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks The speech-recognition mode for a site is saved between uses of the site, even if the user shuts down the computer in the meantime. If a voice-navigation application is installed on the user's computer, an application may not need to set the speech-recognition mode.

An application can use a speech-recognition enumerator to determine which speech-recognition modes are available. For information about the speech-recognition enumerator, see the section, "Speech Recognition."

In Auto PC, there is usually only one speech recognition mode.

IVCmdAttributes::ThresholdGet

IVCmdAttributes::ThresholdGet retrieves the threshold level of the speech-recognition engine used by a voice-command site.

Syntax

```
HRESULT ThresholdGet(
        DWORD *pdwThreshold
);
```

Parameters *pdwThreshold*

[out] Address of a variable that receives the threshold level.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM

Remarks The threshold level is a value from 0 to 100 that indicates the point below which an engine rejects an utterance as unrecognized. A value of 0 indicates that the engine should match any utterance to the closest phrase match. A value of 100 indicates that the engine should be absolutely certain that an utterance is the recognized phrase. For example, suppose the engine is expecting "What is the time?" If the threshold is 100 and the user mumbles "What'z tha time" or has a cold, the command may not be recognized. However, if the threshold is too low and the user says a similar-sounding phrase that is not being listened for such as "What is mine?" the engine may recognize it as "What is the time?"

If the command spoken by the user is not close enough to what the speech-recognition engine expects, the voice-command object notifies the application that the command was not recognized by calling IVCmdNotifySink::CommandOther with a NULL phrase.

The threshold for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

IVCmdAttributes::ThresholdSet

IVCmdAttributes::ThresholdSet sets the threshold level for the speech-recognition engine used by a voice-command site.

Syntax
```
HRESULT ThresholdSet(
        DWORD dwThreshold
);
```

Parameters *dwThreshold*
[in] Threshold level. An application can specify SRATTR_MINTHRESHOLD and SRATTR_MAXTHRESHOLD for minimum and maximum allowable values.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_NOTSUPPORTED
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks The threshold level is a value from 0 to 100 that indicates the point below which an utterance is rejected as unrecognized. A threshold level of 0 indicates that the engine should match any

utterance to the closest phrase match. A value of 100 indicates that the engine should be absolutely certain that an utterance is the recognized phrase. If the value is out of range for the engine, an error is returned and the attribute is not changed.

The threshold for a site is saved between uses of the site, even if the user shuts down the computer in the meantime.

If a voice-navigation application is installed on the user's computer, an application may not need to set the threshold.

IVCmdEnum

The IVCmdEnum interface is a standard OLE enumeration interface. It is used by applications to enumerate the menus stored in the voice-command database.

Method	Description
IVCmdEnum::Clone	Retrieves another enumerator containing the same enumeration state as the current one.
IVCmdEnum::Next	Retrieves the specified number of items in the enumeration sequence.
IVCmdEnum::Reset	Resets the enumeration sequence back to the beginning.
IVCmdEnum::Skip	Skips over a specified number of elements in the enumeration sequence.

Remarks This interface is supported by all voice-command objects.

IVCmdEnum::Clone

IVCmdEnum::Clone retrieves another enumerator containing the same enumeration state as the current one.

Syntax
```
HRESULT Clone(
        IEnumX **ppenum
);
```

Parameters *ppenum*

[out] Address of a variable that receives the cloned enumerator. The type of this parameter is the same as the enumerator name. For example, if the enumerator name is

IEnumFORMATETC, this parameter is of the IEnumFORMATETC type.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- E_OUTOFMEMORY
- E_UNEXPECTED

Remarks Using Clone, it is possible to record a particular point in the enumeration sequence and then return to that point at a later time. The enumerator returned is of the same interface type as the one being cloned.

IVCmdEnum::Next

IVCmdEnum::Next retrieves the specified number of items in the enumeration sequence.

Syntax
```
HRESULT IEnumX::Next(
    ULONG celt,
    Unknown **rgelt,
    ULONG *pceltFetched
);
```

Parameters

celt

[in] Number of elements to retrieve. If the number of elements requested is more than remains in the sequence, only the remaining elements are retrieved.

rgelt

[out] Address of an array that receives the elements. If an error value is returned, no entries in the array are valid.

pceltFetched

[out] Address of a variable that receives the number of array elements actually copied to the array. This parameter cannot be NULL if *celt* is greater than one. If this parameter is NULL, *celt* must be one.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- E_OUTOFMEMORY
- E_UNEXPECTED
- S_FALSE
- S_OK

IVCmdEnum::Reset

IVCmdEnum::Reset resets the enumeration sequence back to the beginning.

Syntax HRESULT IEnumX::Reset(void);

Parameters None

Return Values This method returns NOERROR if successful, or one of these error values:

- S_FALSE
- S_OK

IVCmdEnum::Skip

IVCmdEnum::Skip skips over a specified number of elements in the enumeration sequence.

Syntax
```
HRESULT IEnumX::Skip (
    ULONG celt
);
```

Parameters *celt*

[in] Number of elements to be skipped.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- E_OUTOFMEMORY
- E_UNEXPECTED
- S_FALSE
- S_OK

IVCmdMenu

The IVCmdMenu interface allows an application to manage voice-command menus. It includes methods for such tasks as activating and deactivating menus, and adding and deleting phrases.

Method	Description
IVCmdMenu::Activate	Activates a voice menu so that its commands can be recognized.
IVCmdMenu::Add	Adds one or more commands to a voice menu.

Method	Description
IVCmdMenu::Deactivate	Deactivates an active voice menu.
IVCmdMenu::EnableItem	Permanently enables or disables a menu item.
IVCmdMenu::Get	Retrieves information about one or more commands in a voice menu.
IVCmdMenu::ListGet	Retrieves the phrases stored in the current list for the selected voice menu.
IVCmdMenu::ListSet	Sets the phrases in a list for a voice command.
IVCmdMenu::Num	Retrieves the total number of commands on a voice menu.
IVCmdMenu::Remove	Removes the specified commands from the voice menu.
IVCmdMenu::Set	Sets information for one or more commands in a voice menu.
IVCmdMenu::SetItem	Temporarily enables or disables a command on a voice menu.
IVCmdMenu::TrainMenuDig	Not Implemented

The following flags are used with the member functions of the IVCmdMenu interface to identify a command in a voice-command menu:

VCMD_BY_IDENTIFIER
The *dwCmdNum* is the command identifier of the command.

VCMD_BY_POSITION
The *dwCmdNum* parameter is the position in the list of commands.

Remarks This interface is supported by all voice-command objects.

IVCmdMenu::Activate

IVCmdMenu::Activate activates a voice menu so that its commands can be recognized.

Syntax

```
HRESULT Activate(
    HWND hWndListening,
    DWORD dwFlags
);
```

Parameters *hWndListening*

[in] Handle of the window associated with the voice menu. Whenever this window is the foreground window, the voice menu is automatically activated. Otherwise, it is deactivated. If this parameter is NULL, the voice menu is global (that is, it remains active regardless of the foreground window, until the application explicitly deactivates it).

Note: For the AutoPC, set this parameter to NULL. The application has to activate and deactivate the voice menu manually when the focus switches.

dwFlags

[in] Flag that indicates whether the menu should be active when speech-recognition is "asleep" for the voice-command site. This parameter can be one of these values:

0 or NULL

The voice menu is active only when speech recognition is awake.

VWGFLAG_ASLEEP

The menu is active only when speech recognition is asleep and is automatically deactivated when speech recognition is awake.

Most applications set this parameter to zero. Typically, a sleep menu contains a command to resume speech recognition, such as "Wake up."

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_CANTCREATEDATASTRUCTURES
- VCMDERR_CANTINITDATASTRUCTURES
- VCMDERR_CANTXTRACTWORDS
- VCMDERR_INVALIDWINDOW
- VCMDERR_MENUACTIVE
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_MENUWRONGLANGUAGE
- VCMDERR_NOCACHEDATA
- VCMDERR_NOENGINE
- VCMDERR_NOGRAMMARINTERFACE
- VCMDERR_OUTOFMEM
- VCMDERR_TOOMANYMENUS

Remarks A global voice menu is useful for an application such as a clock program so that the user can ask what time it is and get a response no matter what else he or she is doing. Global voice-menu commands have a lower priority in case of a recognition conflict

– for example, two commands with the same name in different menus.

IVCmdMenu::Add

IVCmdMenu::Add adds one or more commands to a voice menu. The added commands are appended to any existing commands in the menu.

Syntax

```
HRESULT Add(
        DWORD dwCmdNum,
        SDATA dData,
        DWORD *pdwCmdStart
);
```

Parameters

dwCmdNum

[in] Number of commands to add to the menu.

dData

[in] SDATA structure containing a list of VCMDCOMMAND structures that describe the voice commands to be added. Although they vary in size depending on the command data, the structures are contiguous within the list.

pdwCmdStart

[out] Address of a variable that receives the number of the first command added to the menu.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks In Auto PC, applications should use the IAPCSpeech::AddVMenuCommand function in the APC speech interface instead.

Commands are numbered sequentially from 1 to *n*. New commands are added to the end of the menu, so the first command added is numbered *n*+1.

For best results, you should deactivate the voice menu before calling Add. Otherwise, the menu must be deactivated, recompiled, and reactivated before Add returns. If the menu is

already deactivated when Add is called, the menu is not recompiled until the application activates it again.

If a command string includes a list name, you can use IVCmdMenu::ListSet to set the phrases that the user can substitute for the list name when speaking the command.

IVCmdMenu::Deactivate

IVCmdMenu::Deactivate deactivates an active voice menu so that the application no longer listens for its commands.

Syntax HRESULT Deactivate(void);

Parameters None

Return Values This method returns NOERROR if successful, or VCMDERR_OUTOFMEM if a low memory condition exists.

Remarks The menu is still open, so the application can start listening for the menu's commands again by calling IVCmdMenu::Activate to reactive the menu.

IVCmdMenu::EnableItem

IVCmdMenu::EnableItem permanently enables or disables a menu item. When a command is disabled by using EnableItem, it is not compiled into the menu.

Syntax
```
HRESULT EnableItem(
    DWORD dwEnable,
    DWORD dwCmdNum,
    DWORD dwFlag
);
```

Parameters

dwEnable
[in] TRUE to enable the command, or FALSE to disable it.

dwCmdNum
[in] Position or identifier of the command on the menu, depending on the value of *dwFlag*. Command positions are sequential, starting with 1 for the first command on the menu. The command identifier is specified in the dwID member of the VCMDCOMMAND structure that defines the command.

dwFlag

[in] Flag that identifies the nature of *dwCmdNum*. This parameter can be one of these values:

- VCMD_BY_IDENTIFIER
- VCMD_BY_POSITION

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_OUTOFMEM

Remarks For best results, you should deactivate the voice menu before calling EnableItem. Otherwise, the menu must be deactivated, recompiled, and reactivated before the function returns. If the menu is already deactivated when EnableItem is called, the menu is not recompiled until the application activates it again.

IVCmdMenu::Get

IVCmdMenu::Get retrieves information about one or more commands in a voice menu.

Syntax

```
HRESULT Get (
        DWORD dwCmdStart,
        DWORD dwCmdNum,
        DWORD dwFlag,
        PSDATA  pdData,
        DWORD *pdwCmdNum
);
```

Parameters

dwCmdStart

[in] Number of the first command to retrieve. Commands are numbered sequentially from 1 to *n*. If *dwFlag* is the VCMD_BY_IDENTIFIER value, this parameter is ignored.

dwCmdNum

[in] Either the number of commands to retrieve or the identifier of the commands, depending on the value of *dwFlag*. If the sum of *dwCmdStart* and *dwCmdNum* exceeds the total number of commands in the menu, the function returns as many commands as possible.

dwFlag

[in] Flag that identifies the nature of *dwCmdNum*. This parameter can be one of these values:

- VCMD_BY_IDENTIFIER
- VCMD_BY_POSITION

pdData

[out] Address of an SDATA structure that receives the address and size of a buffer. The buffer contains a list of VCMDCOMMAND structures that describe the commands retrieved. Although they vary in size depending on the command data, the structures are contiguous within the list.

pdwCmdNum

[out] Address of a variable that receives the number of commands actually copied to the buffer.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_OUTOFDISK
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks The calling application allocates the SDATA structure and passes its address to Get. Get allocates memory (using the OLE task allocator) for the returned data and sets the pData member of SDATA to point to the memory. If the allocation fails, pData is sent to NULL and the dwSize member is set to zero. The calling application must free the memory pointed to by pData as well as the SDATA structure itself.

The calling application must free the memory allocated by the member function by using the CoTaskMemFree function.

IVCmdMenu::ListGet

IVCmdMenu::ListGet retrieves the phrases stored in the current list for the selected voice menu.

Syntax

```
HRESULT ListGet(
        PTSTR pszList,
        PSDATA  pdList,
        DWORD *pdwListNum
);
```

Parameters *pszList*

[in] Name of the list, such as "name" or "weekday." The list name must appear in the command string for at least one command on the menu. The command string is stored

in the dwCommand member of the VCMDCOMMAND structure that defines the command.

pdList

[out] Address of an SDATA structure that receives the address and size of a buffer. The buffer contains a sequential list of null-terminated strings, one for each phrase in the list.

pdwListNum

[out] Address of a variable that receives the number of phrases that were copied to the buffer. If the list is empty, this parameter receives zero.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDLIST
- VCMDERR_OUTOFMEM

Remarks A list is associated with a menu rather than an individual command. The list must appear in at least one command string, but can be used by more than one command on the menu.

The calling application allocates the SDATA structure and passes its address to ListGet. ListGet allocates memory (using the OLE task allocator) for the returned data and sets the pData member of the SDATA structure to point to the memory. If the allocation fails, the pData member is set to NULL and the dwSize member is set to zero. The calling application must free the memory pointed to by pData, as well as the SDATA structure itself.

It is up to the calling application to free the memory allocated by the member function by using the CoTaskMemFree function.

IVCmdMenu::ListSet

IVCmdMenu::ListSet sets the phrases in a list for a voice command.

Syntax

```
HRESULT ListSet(
        PTSTR pszList,
        DWORD dwListNum,
        SDATA dList
);
```

Parameters *pszList*

[in] Address of the name of the list to set, such as "name" or "weekday." The list name must appear in the command string for at least one command on the menu. The

command string is specified in the dwCommand member of the VCMDCOMMAND structure that defines the command.

dwListNum

[in] Number of phrases in the list.

dList

[in] SDATA structure that contains a pointer to a data buffer and the size of the buffer. The data buffer contains a sequential list of null-terminated strings, one for each phrase in the list.

Returns

This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_INVALIDLIST
- VCMDERR_OUTOFMEM

Remarks

The user can speak any phrase in the list in place of the list name in the command string. A command that uses a list must have the list name in brackets. Example:

"Send mail to <name>"

Calling ListSet establishes a list of phrases that can be spoken in a voice command, such as "Send mail to name." Typically, the list contains information that changes dynamically at run time, such as the ten people to whom the user most recently sent electronic mail. For best results, a list should have fewer than 20 entries. Having more than 20 entries in a list can reduce the accuracy of recognition.

The list persists until the voice-menu object is released. List entries are not automatically saved to disk. To preserve the list, call the IVCmdMenu::ListGet member function and take steps to store the result.

ListSet is much faster than the IVCmdMenu interface's Add, Remove, or Set member functions because list entries are substituted when a command is recognized and the menu is not recompiled. This means that ListSet can be called on an active menu without affecting performance.

IVCmdMenu::Num

IVCmdMenu::Num retrieves the total number of commands on a voice menu.

Syntax

```
HRESULT Num(
        DWORD *pdwNumCmd
);
```

Parameters *pdwNumCmd*

[out] Address of a variable that receives the number of commands.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

IVCmdMenu::Remove

IVCmdMenu::Remove removes the specified commands from the voice menu.

Syntax

```
HRESULT Remove(
        DWORD dwCmdStart,
        DWORD dwCmdNum,
        DWORD dwFlag
);
```

Parameters *dwCmdStart*

[in] Number of the first command in the menu to remove. Command positions are sequential, starting with 1 for the first command on the menu. If *dwFlag* is the VCMD_BY_IDENTIFIER value, this parameter is ignored.

dwCmdNum

[in] Number of commands to remove or the identifier of the commands, depending on the value of *dwFlag*. If the sum of *dwCmdStart* and *dwCmdNum* exceeds the total number of commands in the menu, the function removes as many commands as possible.

dwFlag

[in] Flag that identifies the nature of *dwCmdNum*. This parameter can be one of these values:

- VCMD_BY_IDENTIFIER
- VCMD_BY_POSITION

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_OUTOFDISK
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks For best results, you should deactivate the voice menu before calling Remove. Otherwise, the menu must be deactivated, recompiled, and reactivated before Remove returns. If the menu is already deactivated when Remove is called, the menu is not recompiled until the application activates it again.

IVCmdMenu::Set

IVCmdMenu::Set sets information for one or more commands in a voice menu.

Syntax

```
HRESULT Set(
        DWORD dwCmdStart,
        DWORD dwCmdNum,
        DWORD dwFlag,
        SDATA dData
);
```

Parameters

dwCmdStart

[in] Number of the first command to set in the voice menu. Command positions are sequential, starting with 1 for the first command on the menu. If *dwFlag* is the VCMD_BY_IDENTIFIER value, this parameter is ignored.

dwCmdNum

[in] Either the number of commands to set or the identifier of the commands, depending on the value of *dwFlag*. If the sum of *dwCmdStart* and *dwCmdNum* exceeds the number of commands in the menu, the function sets as many commands as possible.

dwFlag

[in] Flag that identifies the nature of *dwCmdNum*. This parameter can be one of these values:

- VCMD_BY_IDENTIFIER
- VCMD_BY_POSITION

dData

[in] SDATA structure that contains a pointer to a data buffer and the size of the buffer. The data buffer contains

a list of VCMDCOMMAND structures that describe the voice commands to set. Although they vary in size depending on the command data, the structures are contiguous within the list.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUTOOCOMPLEX
- VCMDERR_OUTOFDISK
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks For best results, you should deactivate the voice menu before calling Set. Calling Set on an active menu can be fairly slow because the menu must be deactivated, recompiled, and reactivated before Set returns. If the menu is already deactivated when Set is called, the menu is not recompiled until the application activates it again.

IVCmdMenu::SetItem

IVCmdMenu::SetItem temporarily enables or disables a command on a voice menu.

Syntax

```
HRESULT SetItem(
        DWORD dwEnable,
        DWORD dwCmdNum,
        DWORD dwFlag
);
```

Parameters

dwEnable

[in] TRUE to enable the command or FALSE to disable it.

dwCmdNum

[in] Position or identifier of the command on the menu, depending on the value of *dwFlag*. Command positions are sequential, starting with 1 for the first command on the menu.

dwFlag

[in] Flag that identifies the nature of *dwCmdNum*. This parameter can be one of these values:

- VCMD_BY_IDENTIFIER
- VCMD_BY_POSITION

Return Values This method returns NOERROR, if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_OUTOFMEM

Remarks

If a command is disabled by using SetItem, the voice-command object sends a CommandOther notification rather than a CommandRecognize notification when it "recognizes" the disabled command.

SetItem is much faster than the IVCmdMenu::EnableItem member function because the menu is not recompiled. This means that SetItem can be called on an active menu without affecting performance.

IVCmdNotifySink

The IVCmdNotifySink must be implemented by an application in order to receive notifications from the Voice Command object. In addition to the recognized command, an application can also be notified of events such as: beginning and ending of an utterance, menu activation, and the presence of interference.

Method	Description
IVCmdNotifySink::AttribChanged	A site attribute has changed.
IVCmdNotifySink::CommandOther	A spoken phrase was either recognized as being from another application's command set or was not recognized.
IVCmdNotifySink::CommandRecognize	Recognized as being from the application's command set.
IVCmdNotifySink::CommandStart	A spoken phrase was detected.
IVCmdNotifySink::Interference	Not Implemented
IVCmdNotifySink::MenuActivate	Not Implemented
IVCmdNotifySink::UtteranceBegin	Not Implemented
IVCmdNotifySink::UtteranceEnd	Not Implemented
IVCmdNotifySink::VUMeter	Not Implemented

Remarks

Not all IVCmdNotifySink methods are used by Auto PC SAPI.

IVCmdNotifySink::AttribChanged

IVCmdNotifySink::AttribChanged notifies applications on a voice-command site that a site attribute has changed.

Syntax

```
HRESULT AttribChanged(
    DWORD dwAttribute
);
```

Parameters

dwAttribute

[in] Site attribute that was changed. This parameter can be one of these values:

IVCNSAC_AWAKE
: Awake state.

IVCNSAC_AUTOGAINENABLE
: Automatic gain.

IVCNSAC_DEVICE
: Wave-in audio device.

IVCNSAC_ENABLED
: Enabled state.

IVCNSAC_MICROPHONE
: Current microphone.

IVCNSAC_ORIGINAPP
: The application receiving this notification originated the attribute change.

IVCNSAC_SPEAKER
: Name of the current speaker.

IVCNSAC_SRMODE
: Speech-recognition mode.

IVCNSAC_THRESHOLD
: Confidence threshold.

Return Values The return value is ignored.

Remarks The notification is sent only to applications that, when registered to use voice commands on the site, did one of the following:

- Set the *dwFlags* parameter of the IVoiceCmd::Register member function to the VCMDRF_ALLBUTVUMETER value.
- Set the VCMDRF_ATTRIBCHANGE bit.

dwAttribute includes the IVCNSAC_ORIGINAPP value only if the application sets an attribute by calling the IVCmdAttributes interface's EnabledSet, AwakeStateSet, DeviceSet, or SRModeSet member function.

IVCmdNotifySink::CommandOther

IVCmdNotifySink::CommandOther is sent when a spoken phrase was either recognized as being from another application's command set or was not recognized.

Syntax

```
HRESULT CommandOther(
    PVCMDNAME pName,
    PTSTR pszCommand
);
```

Parameters

pName

[in] Address of a VCMDNAME structure that contains the name of the voice menu that has the recognized command. If this parameter contains NULL, the command was not recognized.

pszCommand

[in] Address of the command string. If this parameter contains NULL, the command was not recognized.

Return Values The return value is ignored.

Remarks

Along with the notification, the application receives the address of the phrase.

An application can use the CommandOther notification to monitor utterances and inform the user what was heard. An application should not rely on this notification for information about the recognition of its own commands. Most applications ignore this notification.

The command string contains the words actually spoken by the user. If the command contains a list name, the command string may not match the words of the command. For example, the string pointed to by *pszCommand* might be "Send mail to Fred" whereas the command string is "Send mail to name."

The notification is sent only to applications that, when registered to use voice commands on the site did one of the following:

- Set the *dwFlags* parameter of the IVoiceCmd::Register member function to the VCMDRF_ALLBUTVUMETER value.
- Set the VCMDRF_CMDOTHER bit.

If two or more voice menus contain the same phrase and this phrase is recognized, it is indeterminate which of the menus will cause the engine to call the IVCmdNotifySink::CommandRecognize notification and which will cause it to call CommandOther. This happens only if the menus are all global or all window specific.

IVCmdNotifySink::CommandRecognize

IVCmdNotifySink::CommandRecognize is sent when a spoken phrase is recognized as being from the application's command set.

Syntax

```
HRESULT CommandRecognize(
        DWORD dwID,
        PVCMDNAME pvCmdName,
        DWORD dwFlags,
        DWORD dwActionSize,
        PVOID pAction,
        DWORD dwNumLists,
        PTSTR pszListValues,
        PTSTR pszCommand
);
```

Parameters

dwID

[in] Identifier of the command that was recognized. The command identifier is stored in the dwID member of the VCMDCOMMAND structure that defines the command.

pvCmdName

[in] Address of a VCMDNAME structure containing the voice menu that has the recognized command.

dwFlags

[in] VCMDCMD_VERIFY if the application should request verification from the user or NULL if verification is not required. To request verification, the application should display a dialog box. An application would typically require verification for a destructive or irreversible command such as "Format disk."

dwActionSize

[in] Size of the data in pAction.

pAction

[in] Address of a string that contains action data to accompany the recognized command. The action data is obtained from the VCMDCOMMAND structure for the command.

dwNumLists

[in] Size, in bytes, of the list data for the command. If a command does not contain any list fields, this parameter is zero.

pszListValues

[in] Address of a list of one or more null-terminated strings that correspond to the phrase from each list in the order that they occur in the command. For example, if the command is "Set the time to number AM or PM," this

parameter points to "Ten\0PM" (the last '\0' is implicit in C notation).

pszCommand

[in] Address of the command string for the command that was recognized.

Return Values The return value is ignored.

Remarks Along with the notification, the application receives the text of the phrase and the action data that was supplied by the application when it originally defined the command.

You should not use the contents of *pszCommand* to identify the recognized command. Instead, use the data in *pAction* or the identifier in *dwID* to determine which command was recognized. The *pszCommand* string may not contain the same string that you specified in the VCMDCOMMAND structure because it is possible for the user to edit the text for commands for your application using Microsoft Voice or another voice-aware application.

The notification is sent to all applications that are registered on the voice-command site, regardless of the settings of the *dwFlags* parameter of the IVoiceCmd::Register member function when the application registered to use voice commands.

If two or more global voice menus (or two or more window-specific voice menus) contain the same phrase and the engine recognizes that phrase, the engine calls CommandRecognize for one menu and IVCmdNotifySink::CommandOther for the other. The engine determines which notification to call for each menu; an application cannot determine which notification will be called.

IVCmdNotifySink::CommandStart

IVCmdNotifySink::CommandStart is sent when a spoken phrase is detected.

Syntax HRESULT CommandStart();

Return Values The return value is ignored.

Remarks The notification is sent only to applications that, when registered to use voice commands on the site, did one of the following:

- Set the *dwFlags* parameter of the IVoiceCmd::Register member function to the VCMDRF_ALLBUTVUMETER value.

- Set the VCMDRF_CMDSTART bit. *dwAttribute* includes the IVCNSAC_ORIGINAPP value only if the application sets an attribute by calling the IVCmdAttributes interface's EnabledSet, AwakeStateSet, DeviceSet, or SRModeSet member function.

IVCmdNotifySink::Interference

IVCmdNotifySink::Interference notifies the application that the engine cannot recognize speech properly for a known reason.

Syntax

```
HRESULT Interference(
    DWORD dwType
);
```

Parameters

dwType

[in] Type of interference. This parameter can be one of these values:

SRMSGINT_AUDIODATA_STARTED
: The engine has resumed receiving audio data from the audio source.

SRMSGINT_AUDIODATA_STOPPED
: The engine has stopped receiving audio data from the audio source.

SRMSGINT_NOISE
: The background noise is too high.

SRMSGINT_NOSIGNAL
: The engine cannot detect a signal, possibly because the microphone is off or unplugged.

SRMSGINT_TOOLOUD
: The speaker is too loud; recognition results may be degraded.

SRMSGINT_TOOQUIET
: The speaker is too quiet; recognition results may be degraded.

Return Values The return value is ignored.

Remarks The notification is sent only to applications that set the *dwFlags* parameter of the IVoiceCmd::Register member function to the VCMDRF_ALLBUTVUMETER value when the application registered to use voice commands on the site.

IVCmdUserWord

The IVCmdUserWord interface allows an application to enable the speaker-dependent and speaker-independent templates, and to add new words to the speaker-dependent template.

Method	Description
IVCmdUserWord::AddRemoveSIFile	Installs or uninstalls speaker-independent template extension files.
IVCmdUserWord::ModifyTraining	Specify which templates are enabled for a particular phrase.
IVCmdUserWord::GetPhraseList	Gets the current phrase list.
IVCmdUserWord::QueryPhrase	Determines what kind of templates a phrase has and whether or not they are enabled.
IVCmdUserWord::Train	Train a list of user-defined phrases.

Remarks

This interface is an extension of the Microsoft Speech API, added to meet the needs of the Auto PC. It is designed specifically for an isolated-word recognizer. Continuous speech recognizers should have training templates for all phrases, and should not need to train user-defined words. Any function call on this interface will affect the current speaker only.

Templates hold information that the engine uses to recognize a phrase. There are two types of templates for the Auto PC: speaker-independent and speaker-dependent. There is one speaker-independent template for each phrase. Each speaker can have one speaker-dependent template for each phrase.

To create a speaker-dependent template, a user must "train" the object to recognize their particular speech pattern. Speaker-independent recognition can only be enabled or disabled. It cannot be modified by the user.

The two templates operate independently of each other. An application can enable a speaker-dependent template whether or not the speaker-independent template is available. Enabling both templates may achieve better recognition accuracy.

IVCmdUserWord::AddRemoveSIFile

The IVCmdUserWord::AddRemoveSIFile method installs or uninstalls speaker-independent template extension files.

Syntax

```
HRESULT AddRemoveSIFILE(
    LPCTSTR    lpszFile,
    BOOL       bInstall);
```

Parameters

lpszFile

Pointer to the path of the file to install or uninstall.

bInstall

Indicates whether to install or uninstall a file, TRUE to install, FALSE to uninstall.

IVCmdUserWord::GetPhraseList

The IVCmdUserWord::GetPhraseList method gets the words in the installed vocabulary.

Syntax

```
HRESULT GetPhraseList(
    DWORD      dwFlags,
    PWSTR      lpBuf
    PDWORD     *pdwByteCnt
);
```

Parameters

dwFlags

There are two flags that can be set, one for each word list. If both are set, the combined list is returned.

Flag	Description
SRPHRASE_SI	Returns the speaker-independent list.
SRPHRASE_SD	Returns the speaker-dependent list.

lpBuf

Pointer to the buffer where the phrase list will be stored.

PdwByteCnt

The size of the buffer allocated to hold the list, in bytes. If the method returns successfully, it holds the actual number of bytes in the buffer.

Return Values This method returns NOERROR if successful, or one of these error values:

VCMDERR_VALUEOUTOFRANGE

The allocated buffer is too small. When this occurs, GetPhraseList will set *pdwByteCnt* to the buffer size needed.

Errors

If there is an error, the appropriate HRESULT should be returned.

Remarks If both of these flags, SRPHRASE_SI and SRPHRASE_SD, are set, and if a word has both speaker-independent and speaker-dependent templates, the same word shows up in the buffer twice.

IVCmdUserWord::ModifyTraining

The IVCmdUserWord::ModifyTraining method allows an application to specify which templates are enabled for a particular phrase.

Syntax

```
HRESULT ModifyTraining(
    LPTSTR lpszPhrase
    DWORD dwFlags
);
```

Parameters

lpszPhrase

The phrase of interest.

dwFlags

SRPHRASE_SI

Specifies the speaker-independent template.

SRPHRASE_SD

Specifies the speaker-dependent template.

SRPHRASE_SI_ENABLE

Enables or disables a phrase on the speaker-independent template.

SRPHRASE_SD_ENABLE

Enables or disables a phrase on the speaker-dependent template.

SRPHRASE_SD_ERASE

Erases the speaker-dependent template for a phrase.

SRPHRASE_PERMANENT

When set, makes any changes permanent.

Return Values This method returns NOERROR if successful, or one of these error values:

SRERR_PHRASENOTFOUND

The phrase was not found in either template.

SRERR_TEMPLATENOTFOUND

The template is not available.

Other Errors

If there is another error, the appropriate HRESULT should be returned.

Remarks Templates are enabled independently of each other. Either or both may be enabled at any given time. When setting a flag to enable or disable a template, the corresponding flag to select the template must also be set. For example, to enable the speaker-dependent template, user SRPHRASE_SD | SRPHRASE_SD_ENABLE.

The phrase string can contain alphabetic characters and intraword punctuation. It may not contain pronounced symbols such as numbers ("345" is not a valid string). Avoid ambiguous pronunciation. Instead of IEEE, use "I triple E," for instance.

IVCmdUserWord::QueryPhrase

The IVCmdUserWord::QueryPhrase method is used to determine what kind of templates a phrase has and whether or not they are enabled.

Syntax
```
HRESULT QueryPhrase(
        LPTSTR      lpszPhrase
        DWORD *pdwValue
);
```

Parameters
lpszPhrase
The phrase of interest.
pdwValue
Returns flags indicating the training templates associated with the phrase.
SRPHRASE_SI
The phrase has a speaker-independent template.
SRPHRASE_SI_ENABLE
The speaker-independent template is enabled/disabled.
SRPHRASE_SD
The phrase has a speaker-dependent template.
SRPHRASE_SD_ENABLE
The speaker-dependent template is enabled/disabled.

Return Values This method returns NOERROR if successful, or one of these error values:
Errors
If there is an error, the appropriate HRESULT should be returned.

Remarks The phrase string can contain alphabetic characters and intraword punctuation. It may not contain pronounced symbols such as

numbers ("345" is not a valid string). Avoid ambiguous pronunciation. Instead of IEEE, use "I triple E," for instance.

IVCmdUserWord::Train

The IVCmdUserWord::Train method is called by the application to train a list of user-defined phrases.

Syntax

```
HRESULT Train(
    LPTSTR lpPhrases
    DWORD dwSize
    DWORD hHandle
    DWORD dwFlags
);
```

Parameters

lpPhrases
A pointer to a sequential list of Unicode text strings. Each string is terminated by a Unicode NULL character. The end of the list is also indicated by a NULL.

dwSize
The number of Unicode characters in the list, including NULL characters (*not the number of bytes!*).

hHandle
Not implemented in AutoPC version 1. This parameter should be set to zero.

dwFlags
Not implemented in AutoPC version 1. This parameter should be set to zero.

Return Values This method returns NOERROR if successful, or one of these error values:

Errors
If there is an error, the appropriate HRESULT should be returned.

Remarks The phrase string can contain alphabetic characters and intraword punctuation. It may not contain pronounced symbols such as numbers ("345" is not a valid string). Avoid ambiguous pronunciation. Instead of IEEE, use "I triple E," for instance.

IVoiceCmd

The IVoiceCmd interface registers an application with a voice-command object. It is also used for tasks such as creating menus and menu enumerators.

Method	Description
IVoiceCmd::CmdMimic	Supplies a voice-aware application with the equivalent of a spoken voice command.
IVoiceCmd::MenuCreate	Creates a voice-menu object.
IVoiceCmd::MenuDelete	Deletes a menu from the voice-menu database.
IVoiceCmd::MenuEnum	Creates a voice-menu enumerator.
IVoiceCmd::Register	Registers an application to use voice commands.

Remarks This interface is supported by all voice-command objects.

IVoiceCmd::CmdMimic

The IVoiceCmd::CmdMimic method supplies a voice-aware application with the equivalent of a spoken voice command.

Syntax

```
HRESULT CmdMimic(
        PVCMDNAME pMenu,
        PTSTR pszCommand
);
```

Parameters

pMenu

[in] Address of a VCMDNAME structure identifying the menu that contains the command to mimic.

pszCommand

[in] Address of a string that contains the command to mimic.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_CANNOTMIMIC
- VCMDERR_INVALIDCHAR
- VCMDERR_MENUDOESNOTEXIST
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE
- VCMDERR_INVALIDCHAR

Remarks CmdMimic parses the command string and eliminates white space and punctuation, and then the member function compares the result with each command on the voice menu until it finds a match. The comparison is case-insensitive, and the command string can include phrases from lists. If the string matches a command in the voice menu, it is recognized. Otherwise, the function returns an error.

An application can call CmdMimic to play back voice macros to another application, somewhat like playing back keystrokes and mouse messages in Windows.

The voice menu must be active before an application can mimic its commands.

IVoiceCmd::MenuCreate

The IVoiceCmd::MenuCreate method creates a voice-menu object to represent a new or existing voice menu for an application.

Syntax

```
HRESULT MenuCreate(
    PVCMDNAME pName,
    PLANGUAGE pLanguage,
    DWORD dwFlags,
    PIVCMDMENU *ppIVCmdMenu
);
```

Parameters

pName

[in] Address of a VCMDNAME structure that identifies the menu to create. The VCMDNAME structure contains an application name, such as "Excel," and a state name, such as "Main menu" or "File Open dialog box."

pLanguage

[in] Address of a LANGUAGE structure that indicates the language to use for the menu. If this parameter is NULL, the default language for the site's speech-recognition mode is used.

dwFlags

[in] Flag that indicates how to create the menu. This parameter can be one of these values:

VCMDMC_CREATE_ALWAYS

Creates an empty menu with the given name. If a menu by that name already exists in the voice-menu database, it is erased. The new menu is stored in the database when the menu object is released.

VCMDMC_CREATE_NEW

Creates an empty menu with the given name. If a menu by that name already exists in the voice-menu database, the function returns an error. The new menu is stored in the database when the menu object is released.

VCMDMC_CREATE_TEMP
Creates an empty menu with the given name. If a menu by that name already exists in the voice-menu database, the function returns an error. The new menu is temporary and is discarded when the menu object is released.

VCMDMC_OPEN_ALWAYS
Opens an existing menu with the given name. If the menu does not exist, the function creates a new, empty menu. The new menu is stored in the database when the menu object is released.

VCMDMC_OPEN_EXISTING
Opens an existing menu. If the menu does not exist, the function returns an error.

ppIVCmdMenu
[out] Address of an IVCmdMenu interface for the newly created voice-menu object. The application uses the pointer to this interface to call IVCmdMenu functions on the voice-menu object. If an error occurs, this parameter receives NULL.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_CANTCREATESTORAGE
- VCMDERR_MENUDOESNOTEXIST
- VCMDERR_MENUEXIST
- VCMDERR_OUTOFDISK
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE

Remarks An application can create a voice-menu object by loading an existing voice menu from the voice-menu database or creating a new voice menu. A voice menu need not be stored in the database; an application can create a temporary voice menu by setting *dwFlags* to the VCMDMC_CREATE_TEMP value. A temporary voice menu persists until the menu object is released.

An application can create more than one voice-menu object to represent the same menu — either one of its own menus or a menu for another application. For example, one application might do this to enumerate another application's menus.

More than one application can use the same voice-menu object. For example, Application A might call the IVoiceCmd::CmdMimic member function on a voice-menu object that represents a menu for Application B, while

Application B uses the same menu object to recognize commands spoken by the user.

IVoiceCmd::MenuDelete

The IVoiceCmd::MenuDelete method deletes a menu from the voice-menu database.

Syntax

```
HRESULT MenuDelete(
    PVCMDNAME pName
);
```

Parameters *pName*

[in] Address of a VCMDNAME structure that identifies the menu to delete.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_MENUACTIVATE
- VCMDERR_MENUDOESNOTEXIST
- VCMDERR_MENUOPEN
- VCMDERR_OUTOFMEM

Remarks A menu cannot be deleted if it is currently open and the application is actively listening for its commands.

This function deletes the storage in the database for the menu (if it exists) and releases the voice-menu object that was created by the IVoiceCmd::MenuCreate member function. After a menu is deleted, the pointer to its IVCmdMenu interface is invalid, so it should be set to NULL.

IVoiceCmd::MenuEnum

The IVoiceCmd::MenuEnum method creates a voice-menu enumerator that allows an application to enumerate menus in the voice-menu database.

Syntax

```
HRESULT MenuEnum(
    DWORD dwFlags,
    PTSTR pszApplicationFilter,
    PTSTR pszStateFilter,
    PIVCMDENUM *ppiVCmdEnum
);
```

Parameters *dwFlags*

[in] Indicates whether to enumerate active menus or open menus (those that have voice-menu objects, whether or not they are also active). This parameter can be certain combinations of these values:

VCMDEF_ACTIVE
Enumerates only active menus.

VCMDEF_DATABASE
Enumerates all menus in the voice commands database.

VCMDEF_PERMANENT
Enumerates only permanent menus.

VCMDEF_SELECTED
Enumerates open menus, whether or not they are also active.

VCMDEF_TEMPORARY
Enumerates only temporary menus.

VCMDEF_ACTIVE and VCMDEF_SELECTED are mutually exclusive, as are VCMDEF_TEMPORARY and VCMDEF_PERMANENT. If both values are specified, the function returns an error. VCMDEF_TEMPORARY and VCMDEF_PERMANENT are ignored if neither VCMDEF_ACTIVE and VCMDEF_SELECTED are specified. In other words, they do not apply if you want to enumerate the menus in the database. By definition, if a menu is active, it is selected.

pszApplicationFilter

[in] Address of the name of the application for which to enumerate menus. This name is the same as that in the szApplication member of the VCMDNAME structure passed to the IVoiceCmd::MenuCreate member function. If this parameter is NULL, menus for all applications, except those eliminated by *dwFlags* and *pszStateFilter*, are enumerated.

pszStateFilter

[in] Address of a string that contains the name of the state for which to enumerate menus. This is the same as in the szState member of the VCMDNAME structure passed to MenuCreate. If *pszApplicationFilter* is NULL, all menus except those eliminated by *dwFlags* and this parameter are enumerated.

ppiVCmdEnum

[out] Address of a variable that receives a pointer to an IVCmdEnum interface for the newly created voice-menu enumerator. If an error occurs, this parameter receives NULL.

Return Values This method returns NOERROR if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_INVALIDMODE
- VCMDERR_OUTOFMEM
- VCMDERR_VALUEOUTOFRANGE
- VCMDERR_MENUDOESNOTEXIST

Remarks A menu can use a voice-menu enumerator to find and modify unknown menus or to show menu status to the user.

The voice-menu enumerator persists until all references to it are released, even if the voice-command object is released.

IVoiceCmd::Register

The IVoiceCmd::Register method registers an application to use voice commands on a site. An application must call this function before it can use voice commands.

Syntax

```
HRESULT Register(
    PTSTR pszSite,
    PIVCMDNOTIFYSINK pNotifyInterface,
    IID IIDNotifyInterface,
    DWORD dwFlags,
    PVCSITEINFO pSiteInfo
);
```

Parameters

pszSite

In Auto PC, must be null or empty.

pNotifyInterface

[in] Address of the notification interface that receives notifications from the voice-command object. The interface identifier is specified by IIDNotifyInterface. If this parameter is NULL, no notifications will be sent.

Because passing the pointer to the voice-command object does not transfer ownership of the notification interface, the voice-command object must call the AddRef member function of the notification interface before returning from the call to Register. The voice-command object must also call the Release member function of the notification interface when it closes. The calling application must release any reference counts it holds on the notification interface after calling Register, unless it needs the notification object to be valid when the voice-command object releases it.

IIDNotifyInterface

[in] GUID that uniquely identifies the type of notification sink being passed to the voice-command object. It must be IID_IVCmdNotifySinkW.

dwFlags

[in] Flag that indicates whether the application is to receive all notifications. This parameter can be one of these values:

VCMDRF_ALLMESSAGES
Sends all notifications to pNotifyInterface.

VCMDRF_ALLBUTVUMETER
Sends all but VUMeter notifications to pNotifyInterface.

VCMDRF_VUMETER
Sends VUMeter notifications to pNotifyInterface.

VCMDRF_NOMESSAGES
Does not send notifications.

If *dwFlags* is 0 (zero) or NULL, only the IVCmdNotifySink::CommandRecognize notification is sent.

pSiteInfo

[in] Address of a VCSITEINFO structure that contains settings to apply to the site, such as the speaker, confidence threshold, and speech-recognition mode. The settings are applied even if the site is already open. If this parameter is NULL, the voice-command object uses the settings from the registry. If there are no registry settings, it uses the default settings, typically those for the computer.

Telephony applications will pass this information to ensure that the proper settings are selected. Other applications will set this parameter to NULL to leave the site settings unchanged from previous values.

Return Values This method returns NOERROR, if successful, or one of these error values:

- E_INVALIDARG
- VCMDERR_CANTCREATEAUDIODEVICE
- VCMDERR_CANTCREATESRENUM
- VCMDERR_CANTSELECTENGINE
- VCMDERR_CANTSETDEVICE
- VCMDERR_INVALIDMODE
- VCMDERR_NOFINDINTERFACE
- VCMDERR_NOSITEINFO
- VCMDERR_OUTOFMEM
- VCMDERR_SRFINDFAILED
- VCMDERR_VALUEOUTOFRANGE

Remarks

An application cannot call Register a second time for the same voice-command object. If a voice-command object is already registered, calling Register returns an error. To change sites, the application must call CoCreateInstance to create a new voice-command object for the desired site.

An application must call Register before it can call any of the following member functions:

See Also

IVCmdMenu::Deactivate, IVCmdMenu::ListGet, IVCmdMenu::ListSet

Detailed Description of Data Structures for a Voice Command API

Chapter 24

VCMDCOMMAND

Provides information about a command in a voice menu.

```
typedef struct {  // vccmd
        DWORD     dwSize;
        DWORD     dwFlags;
        DWORD     dwID;
        DWORD     dwCommand;
        DWORD     dwDescription;
        DWORD     dwCategory;
        DWORD     dwCommandText;
        DWORD     dwAction;
        DWORD     dwActionSize;
        BYTE      abData[];
} VCMDCOMMAND, *PVCMDCOMMAND;
```

Members

dwSize

Size, in bytes, of the VCMDCOMMAND structure, including the amount allocated for abData. The contents of abData must be doubleword-aligned, so round dwSize up to the nearest whole doubleword.

dwFlags

Flags that indicate information about the command. This member can be a combination of these values:

Value	Description
VCMDCMD_DISABLED_PERM	The command was disabled by using the IVCmdMenu::EnableItem member function so that voice commands will not recognize it. The command is not compiled into the voice menu.
VCMDCMD_DISABLED_TEMP	The command was disabled by using the IVCmdMenu::SetItem member function. The command is still compiled into the voice menu, however, so it can be re-enabled without recompilation of the menu.

Value	Description
VCMDCMD_VERIFY	The application should prompt the user to verify the command before carrying it out. For example, this value should be set for a "Delete File" command. This value may be combined with either of the other values.
VCMDCMD_CANTRENAME	(New for 3.0). This indicates that the command is automatically generated and that navigator applications (such as Microsoft Voice) shouldn't allow users to rename the command. For example: A set of commands that are generated by extracting all of the menu items in the currently running application would have this flag set since there would be little point in users renaming them.

dwID

Command identifier. This member can be used to identify a command to modify, or it can be used for notifications.

dwCommand

Offset from the beginning of this structure to first character of the voice command string (ANSI or Unicode, depending on which character set was used in the application). For example, the voice command string might be "Open the file" and the character at the offset specified by dwCommand would be 'O'. In languages such as Japanese that have both a phonemic and symbolic character set, the dwCommand member is the offset to a phonemic string.

Within the command string, the following characters have special meaning:

Character	Meaning
	Indicates the name of a list of words or phrases that can be spoken at this point in the command. For example, the command string "Send mail to name" contains a list called "name." To add phrases to the list, use the IVCmdMenu::ListSet member function.
{ }	Reserved for future use.
[ ]	Reserved for future use.

dwDescription

Offset from the beginning of the structure to first character of a string that describes the action performed by the command.

dwCategory

Offset from the beginning of the structure to the first character of a string that indicates the category to which the command belongs.

Commands in a voice menu should be organized in different categories to help the user browse through lists of commands more easily. This is similar in concept to Windows menus, which organize commands under menu names such as "File," "Edit," "View," and so on. For best results, you should use 20 or fewer categories.

dwCommandText

Offset from the beginning of the structure to the first character of the command text, which is the string that is displayed to the user when he or she requests a list of available voice commands. If this member is NULL, an application uses the text pointed to by dwCommand, which is the voice-command string used in the application's user interface.

Most applications written for European languages will set this member to NULL because the language uses only one character set. Applications written for languages that have both a phonemic and symbolic character set, such as Japanese, will store the phonemic representation of the command in dwCommand and the symbolic representation (which is preferred by the user) in this member.

dwAction

Offset from the beginning of the structure to the first byte of a block of data that is sent to the application when the command is spoken.

Data passed with a command is not interpreted by voice commands; it is up to the application to determine whether the data is valid and to act upon it. Always check the validity of the data, because it is susceptible to being changed — accidentally or intentionally — by other

applications, just as other applications can change an .INI file or registry file.

dwActionSize

Number of bytes required to store the block of data indicated by dwAction.

abData

Array of type BYTE that contains the command string, its description, its category, and additional data (if any) to pass to the application along with the command. Because all of the items in abData are doubleword-aligned, the size of abData should be a multiple of 4. All strings should be null-terminated (\0).

Because of the items indicated by offsets into abData are doubleword-aligned, the offsets specified by dwCommand, dwDescription, dwCategory, dwAction, and dwActionSize should be multiples of 4.

VCMDNAME

Contains strings that uniquely identify the application and state to which a voice menu belongs.

```
typedef struct {  //  vcn
        TCHAR       szApplication[VCMD_APPLEN];
        TCHAR       szState[VCMD_STATELEN];
} VCMDNAME,      *PVCMDNAME;
```

szApplication

Name of the application — for example, "Microsoft Word." The application name must be unique among all applications registered to use voice commands on the user's computer.

szState

Unique name of the application state in which the voice command set is valid. An application state usually corresponds to an active window or dialog box — for example, "Main Window" or "Set Font Dialog."

VCSITEINFO

Provides information about the audio device, speech-recognition mode, and other attributes of a voice-command site.

```
typedef struct {   // vcsi
        DWORD       dwAutoGainEnable;
        DWORD       dwAwakeState;
        DWORD       dwThreshold;
```

```
    DWORD     dwDevice;
    DWORD     dwEnable;
    TCHAR     szMicrophone[VCMD_MICLEN];
    TCHAR     szSpeaker[VCMD_SPEAKERLEN];
    GUID      gModeID;
} VCSITEINFO,     *PVCSITEINFO
```

dwAutoGainEnable
: Value from 0 to 100 that indicates the state of the automatic gain for the incoming audio stream to be used by the site.

dwAwakeState
: TRUE if the site is awake for purposes of speech recognition or FALSE if the site is asleep.

dwThreshold
: Value from 0 to 100 that indicates the recognition threshold for the speech-recognition engine to be used by the site.

dwDevice
: Device identifier of the wave-in audio device to be used by the site. The device identifier can be obtained by calling the waveInGetNumDevs and waveInGetDevCaps multimedia functions.

dwEnable
: TRUE if speech-recognition is enabled for the site or FALSE if speech-recognition is disabled.

szMicrophone
: Name of the current microphone for the audio source to be used by the site.

szSpeaker
: Name of the current speaker for the site.

gModeID
: GUID that uniquely identifies the speech-recognition mode to be used by the site. The GUID for a speech-recognition mode can be obtained by using a speech-recognition enumerator. For more information about speech-recognition enumerators, see section, "Low-Level Speech Recognition API."

Remarks

An application can pass a pointer to a VCSITEINFO structure with the IVoiceCmd::Register function to set the audio device, speech-recognition mode, and other attributes of a voice-command site, even if the site is already open.

Chapter 25

VTSITEINFO

Specifies an audio device, a text-to-speech mode, and the talking speed for a voice-text site and indicates whether voice text is enabled or disabled for the site.

```
typedef struct {    // vtsi
        DWORD       dwDevice;
        DWORD       dwEnable;
        DWORD       dwSpeed;
        GUIDgModeID;
} VTSITEINFO,       *PVTSITEINFO;
```

Members

dwDevice

Device identifier of the wave-out audio device to be used by the site. The device identifier can be obtained by calling the waveOutGetNumDevs and waveOutGetDevCaps multimedia functions.

dwEnable

TRUE if voice text is to be enabled for the site or FALSE if voice text is to be disabled.

dwSpeed

Baseline average talking speed, in words per minute, for the text-to-speech mode to be used by the site.

gModeID

GUID that uniquely identifies the text-to-speech mode to be used by the site. The GUID for a text-to-speech mode is obtained from a text-to-speech enumerator object. For information about text-to-speech enumerators, see the section, "Low-Level Text-to-Speech API."

An application can specify the address of a VTSITEINFO structure in a call to the IVoiceText::Register member function to set the voice, speaking speed, and other attributes of a voice-text site, even if the site is already open. Telephony applications typically do this to ensure that the proper information is selected for the site.

Detailed Description of a Voice Command API for an Auto PC

Chapter 2

IAPCSpeech

The IAPCSpeech interface is a high level Auto PC simple speech interface.

Remarks The function CreateAPCSpeechObject should be called to get the IAPCSpeech interface because APCSpeechObj cannot be created using CoCreateInstance.

IAPCSpeech Methods

Methods	Description
IAPCSpeech::AddRefwscesdk_IAPCSpeech_AddRef	Increments the reference count for an interface on a speech object.
IAPCSpeech::AddVMenuCommandwcesdk_IAPCSpeech_AddVMenuCommand	Adds a command to the voice menu *pmenu*.
IAPCSpeech::AttribGetwcesdk_IAPCSpeech_AttribGet	Gets speech-related settings.
IAPCSpeech::AttribSetwcesdk_IAPCSpeech_AttribSet	Sets or changes speech-related settings.
IAPCSpeech::CreateVMenuwcesdk_IAPCSpeech_CreateVMenu	Creates a voice menu.
IAPCSpeech::QueryInterfacewcesdk_IAPCSpeech_QueryInterface	Returns a pointer to an IAPCSpeech interface.
IAPCSpeech::Releasewcesdk_IAPCSpeech_Release	Decrements the reference count.
IAPCSpeech::Speakwcesdk_IAPCSpeech_Speak	Says or speaks the string stored in *szTTS* using TTS.
IAPCSpeech::Trainwcesdk_IAPCSpeech_Train	Trains the application to recognize a user command.
IAPCSpeech::VoiceHelpStartwcesdk_IAPCSpeech_VoiceHelpStart	Is called by the shell to start voice help.
IAPCSpeech::VoiceHelpStopwcesdk_IAPCSpeech_VoiceHelpStop	Is called by the shell to stop voice help.

IAPCSpeech::AddRef

The IAPCSpeech::AddRef method increments the reference count for an interface on a speech object.

Syntax STDMETHOD_(ULONG) IAPCSpeech::AddRef(THIS) PURE;

IAPCSpeech::AddVMenuCommand

IAPCSpeech::AddVMenuCommand adds a command to the voice menu *pMenu.*

Syntax

STDMETHOD IAPCSpeech::AddVMenuCommand(THIS_
PIVCMDMENUW *pMenu,*
LPTSTR *szCmdString,*
UINT *dwCmdID,*
DWORD *dwFlags,*
PVOID *p*) PURE;

Parameters

pMenu
Pointer to the menu to which a command is to be added.

szCmdStringr
The command string that is to be added to *pMenu.*

dwCmdID
The command ID that is to be added to the voice menu. See Remarks.

dwFlags
Usually set to 0 to allow the system to handle the feedback. If the application wants to control feedback, it can pass:
_none Application handles the feedback tone.
_tone Feedback is always tone.
_echo Feedback is always echo.

p
Must be NULL.

Remarks

To avoid string ID duplication, if your application uses speech-enabled controls, make sure you use the following ranges to assign IDs in resource file:

- Application 0 to 0x7FFF
- Speech enabled controls 0x8000 to 0xFFFF.

IAPCSpeech::AttribGet

IAPCSpeech::AttribGet gets speech-related settings.

Syntax

STDMETHOD IAPCSpeech::AttribGet(THIS_DWORD *dwAttrib,* PDWORD *pdwMisc*) PURE;

Remarks

AttribGet and AttribSet are now called by the shell and the control panel applications. Your application should not call them at the present time.

IAPCSpeech::AttribSet

IAPCSpeech::AttribSet sets or changes speech-related settings.

Syntax STDMETHOD IAPCSpeech::AttribGet(THIS_DWORD *dwAttrib*, DWORD *dwMisc*) PURE;

Remarks AttribGet and AttribSet are now called by the shell and the control panel applications. Your application should not call them at the present time.

IAPCSpeech::CreateVMenu

IAPCSpeech::CreateVMenu creates a voice menu.

STDMETHODIAPCSpeech::CreateVMenu
(THIS_PIVOICECMDW *pVCmd*,
LPCTSTR *lpMenuName*
HINSTANCE *hInst*
DWORD *dwCmdCnt*
LPVOID *pCmdTable*
DWORD *dwFlags*
PIVCMDMENUW* *ppVMenu*) PURE;

Parameters *pVCmd*
Pointer to a voice command. Usually an application should pass null, unless it creates the voice command.
lpMenuName
Unique menu name for each Apcspch object.
hInst
Application or dynamic link library instance handle.
dwCmdCnt
Table size.
pCmdTable
Points to a GrammarID table which stores the resource string ID.
dwFlags
Must be set to 0 or flag listed below. (See Remarks.)
ppVMenu
Pointer to a voice menu pointer.

Remarks 1. *dwFlags*
APCSPCH_VM_USEEXISTING
- The APCSPCH_VM_USEEXISTING flag can be passed in the *dwFlags* parameter. When APCSPCH_VM_USEEXISTING is set and the application finds that the menu already exists, it will use the menu stored in the storage file. You can still pass in

the string table pointer and it is ignored if the APCSPCH_VM_USEEXISTING flag is set and there are commands in the menu.

- NOTE: APCSPCH_VM_USEEXISTING applies only to the CreateVMenu function. A developer should be careful about using AddVMenuCommand while using the APCSPCH_VM_USEEXISTING flag and CreateVMenu to create a voice menu. AddVMenuCommand does not check to determine whether the command is already stored or not. Make sure that you do not add the same command twice.

2. The caller is responsible for releasing the menu object by calling Release. To create a menu in the default voice command *pVCmd* should be NULL. If the application has another voice command, it can pass it to *pVCmd*.
3. The application should call the Activate and Deactivate functions of the menu object to activate or deactivate the grammar.

IAPCSpeech::QueryInterface

IAPCSpeech::QueryInterface returns a pointer to an IAPCSpeech interface.

STDMETHOD IAPCSpeech::QueryInterface(THIS_REFIID *riid*, LPVOID FAR* *ppvObj*) PURE;

Parameters *riid*
[in] Specifies the IID of the interface being requested.
ppvObj
[out] Receives a pointer to an interface pointer to the object on return. If the interface specified in iid is not supported by the object, ppvObject is set to NULL.

Remarks The application can call QueryInterface to obtain the IID_IVoiceCmd, IID_IVoiceText, and other related VoiceCmd and VoiceText interface pointers.

IAPCSpeech::Release

The IAPCSpeech::Release method decrements the reference count for the calling interface on a speech object.

STDMETHOD_(ULONG) IAPCSpeech::Release(THIS) PURE;

IAPCSpeech::Speak

IAPCSpeech::Speak says or speaks the string stored in *szTTS* using TTS.

STDMETHOD IAPCSpeech::Speak(THIS_WCHAR* *szTTS*, DWORD *dwID*) PURE;

Parameters *szTTS*
String that is to be said or spoken.
wID
String buffer ID.

Remarks If the parameter is null, it stops the speech output.

IAPCSpeech::Train

IAPCSpeech::Train trains the application to recognize a user command. It deals with only one command at a time. The function pops up a training form to help the user train the application to recognize a word or command. The function is blocked until the training is finished or cancelled.

STDMETHOD IAPCSpeech::Train(THIS_BSTR *bstrPhrase*, PVOID *pFormManager*) PURE;

Parameters *bstrPhrase*
The word being trained.
pFormManager
Pointer to the application form manager.

IAPCSpeech::VoiceHelpStart

IAPCSpeech::VoiceHelpStart is called by the shell to start voice help.

STDMETHOD IAPCSpeech::VoiceHelpStart(THIS_DWORD *promptType*)PURE;

Parameters *promptType*
Reserved. Must be 0.

Remarks The application should not call VoiceHelpStart or VoiceHelpStop.

IAPCSpeech::VoiceHelpStop

IAPCSpeech::VoiceHelpStop is called by the shell to stop voice help.

STDMETHOD IAPCSpeech::VoiceHelpStop(THIS_DWORD *dwReserved*)PURE;

Parameters *dwReserved*
Reserved. Must be 0.

Remarks Your application must not call VoiceHelpStart or VoiceHelpStop.

CreateAPCSpeechObject

CreateAPCSpeechObject creates an Auto PC speech object.

Syntax CAPCSpeech* CreateAPCSpeechObject(LPCTSTR *lpName*,
DWORD *dwNotifyID*,
DWORD *dwFlags*,
DWORD *dwVCmdOption*,
DWORD *dwTxtOption*);

Parameters Note: At this writing you may use either the thread method or sink method to create a speech object, however, in the future only the sink method may be supported. If your application uses a control that has the speech enabled such as an edit control or an HTML control, you must create the application using the sink method.

lpName
A unique name, usually the application name.

dwNotifyID
Thread Method: The thread ID where the notification messages are posted. Sink Method: The form manager pointer.

dwFlags
Thread Method: Must be 0. Sink Method: Should be APCSPCH_CB_FORMSINK.

dwVCmdOption
This should be set to 0 if the caller is only interested in the recognition notification WM_SPCH_RECOG. It can also be combinations of the following flags: VCMDRF_CMDOTHER, VCMDRF_CMDSTART, VCMDRF_ATTRIBCHANGE.

dwTxtOption
This can be a combination of the following flags: VTXTF_SPEAK, VTXTF_SPEAKDONE, VTXTF_SPEAKSTOP, VTXTF_SPEAKSTART.

Remarks

1. To avoid string ID duplication, if your application uses speech-enabled controls, make sure you use the following ranges to assign string IDs in resource file:
   - Application 0 to 0x7FFF.
   - Speech-enabled controls 0x8000 to 0xFFFF.
2. An application can embed "\mrk=xx\" strings inside the text to be spoken. When the speech engine encounters the bookmarks, a WM_SPCH_NOTIFY (wParam=VTXTF_SPEAK, lParam=bookmarkID) message will be posted to the application. The traditional Speak(*string,ID*) will also work because the system adds \mrk=ID\ before the string and then sends it to the engine.

Detailed Description of an Out-of-Memory API

Chapter 29

Out of Memory User Interface Reference

The out of memory component (Oomui) is a replaceable component that defines the behavior of the Windows CE operating system, including various dialogs and messages, when an out of memory situation is detected.

If you choose to replace the out of memory component with a customized out of memory component, you must implement all of the functions described in this section. Microsoft can provide assistance in this effort, in the form of sample code, upon request.

OomUI_CreateNotRespondingWindow

The OomUI_CreateNotRespondingWindow function creates and returns a handle to a message dialog indicating that an application is not responding.

Syntax HWND OomUI_CreateNotRespondingWindow(void)

Parameters None.

Return Value Handle to the created window.

Remarks The OomUI_CreateNotRespondingWindow function creates and returns a handle to an Application Not Responding dialog. This dialog is displayed if the out of memory component is unable to close a running application. The Out of Memory component should not destroy or hide this window. This function is declared in the header file oomui.h.

OomUI_CreateOomWindow

The OomUI_CreateOomWindow function creates the Out of Memory dialog.

Syntax HWND OomUI_CreateOomWindow(void);

Parameters None.

Return Value Returns a handle to the created window.

Remarks Creates and returns a handle to the Out of Memory dialog. The Out of Memory dialog is immortal, meaning that it should not be destroyed or hidden by the Out of Memory component. This function is declared in the header file oomui.h.

OomUI_FShowOomWindow

The OomUI_FShowOomWindow function is called when the system determines that the Out of Memory window should be shown. It does not display the dialog, however.

Syntax BOOL OomUI_FShowOomWindow(void)

Parameters None.

Return Value Returns TRUE if the Out of Memory window should be shown; otherwise, FALSE.

Remarks This function gives the Out of Memory component a chance to prevent the Out of Memory dialog from appearing (by returning FALSE). This is not recommended, however, unless there are no options available to the user to free more memory. This function is declared in the header file oomui.h.

OomUI_Initialize

The OomUI_Initialize function is called once to initialize the Out of Memory user interface component.

Syntax
```
VOID OomUI_Initialize(
        HINSTANCE hinst
);
```

Parameters *hinst*
The HINSTANCE to use for loading resources.

Return Value None.

Remarks This function is called only once. It gives the Out of Memory user interface component an opportunity to do whatever initialization is needed. This function also informs the Out of Memory component of the current HINSTANCE, which is used to load resources. This function is declared in the header file oomui.h.

OomUI_NotRespondingWndProc

The window procedure for the Not Responding dialog.

Syntax LRESULT CALLBACK OomUI_OomWndProc(
HWND *hwnd*,
UINT *message*,
WPARAM *wParam*,
LPARAM *lParam*
);

Parameters *hwnd*
Handle to the Application Not Responding dialog.
message
A windows message (e.g., WM_CLOSE).
wParam
Message-specific parameter.
lParam
Message-specific parameter.

Remarks This function is the window procedure for the Application Not Responding window. This function is declared in the header file oomui.h.

OomUI_OnShow

The OomUI_OnShow function is called just prior to the showing of the Out of Memory window.

Syntax VOID OomUI_OnShow(void))

Parameters None.

Return Value None.

Remarks The OomUI_OnShow function is called just before the Out of Memory dialog is shown. The OomUI_OnShow function is not required to do anything, but may be used to, for example, set the title of the Out of Memory dialog or collect system information to be displayed in the Out of Memory dialog. This function is declared in the header file oomui.h.

OomUI_OomWndProc

The window procedure for the Out of Memory dialog.

Syntax LRESULT CALLBACK OomUI_OomWndProc(
HWND *hwnd*,
UINT *message*,
WPARAM *wParam*,

LPARAM *lParam*
);

Parameters *hwnd*
Handle to the Out of Memory window.
message
A message (e.g., WM_CLOSE).
wParam
Message-specific parameter.
lParam
Message-specific parameter.

Remarks This function is the window procedure for the Out of Memory window. This function is declared in the header file Oomui.h.

OomUI_SetWindowsInfo

The OomUI_SetWindowsInfo function provides the Out of Memory component with information regarding the windows to be closed.

Syntax VOID OomUI_SetWindowsInfo(
INT *cWindows;*
WINDOWINFO* *rgwi*
);

Parameters *cWindows*
Number of entries in the *rgwi* array.
rgwi
Array of WINDOWINFO structures.

Return Value None.

Remarks The OomUI_SetWindowsInfo function specifies to the Out of Memory component the windows to be closed. Each window is represented as a WINDOWINFO structure. This function and the WINDOWINFO structure are declared in the header file oomui.h.

See Also WINDOWINFO

OomUICallBack_CloseWindow

The OomUICallback_CloseWindow function attempts to close a window.

Syntax
```
BOOL OomUICallback_CloseWindow(
    WINDOWINFO* pwi
);
```

Parameters *pwi*

Pointer to a WINDOWINFO structure.

Return Value Returns TRUE if WM_CLOSE was sent; otherwise FALSE.

Remarks The OomUICallback_CloseWindow function is called by the Out of Memory component, and indicates that the Out of Memory component is attempting to close a window (via WM_CLOSE). If this function returns FALSE, the window could not be sent a WM_CLOSE. If the function returns TRUE, it was sent a WM_CLOSE message. Note that a TRUE return value does not indicate whether the specified window was actually closed.

For more information, see Sample Serial Port Driver.

OomUICallback_IsCritical

The OomUICallback_IsCritical function is called by the Out of Memory component to determine if memory is critically low.

Syntax
```
BOOL OomUICallback_IsCritical(void)
```

Parameters None.

Return Value None.

Remarks The OomUICallback_IsCritical function is called by the Out of Memory component to determine if memory is critically low. This function is declared in the header file Oomui.h.

OomUICallback_NonClientPaint

The OomUICallback_NonClientPaint function is called by the Out of Memory component to paint its non-client area in the "active" colors.

Syntax
```
VOID OomUICallback_NonClientPaint(
    HWND hwnd
);
```

Parameters *hwnd*

Handle to the window.

Return Value None.

Remarks The OomUICallback_NonClientPaint function causes the non-client area (the title bar) to be painted in its "active" color. This function is declared in the header file Oomui.h.

WINDOWINFO

The WINDOWINFO structure defines the window handle, window name, and close options for a window.

Syntax
```
typedef struct {
        HWND hwnd;
        LPCTSTR szWindowName;
        UINT32 fToBeClosed;
        UINT32 fToBeTerminated;
} WINDOWINFO;
```

Members

hwnd
Handle to the window.

szWindowName
Title of the window.

fToBeClosed
A value of 1 indicates that the window should be closed.

fToBeTerminated
A value of 1 indicates that the window should be terminated.

Remarks The WINDOWINFO structure supports the implementation of the Out of Memory component. This structure is declared in header file Oomui.h.

See Also OomUI_SetWindowsInfo, OomUI_SetWindowsInfo, OomUICallback_CloseWindow.

Conclusion

APIs for modules and components of a resource-limited operating system have been described. The APIs provide access to specialized hardware and software that is desirable in such limited-resource systems.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention.

For example, those of ordinary skill within the art will appreciate that while the embodiments of the invention have been described as being implemented in a resource-limited environment, the principles of the invention are applicable to other environments. For example, the voice command APIs can be adapted to standard desk-top operating system to aid user's who have difficulty using a conventional keyboard and mouse to provide input to a system.

The terminology used in this application is meant to include all of these environments. Therefore, it is manifestly intended that this invention be limited only by the following claims and equivalents thereof.

What is claimed is:

1. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, said operating system having a handwriting recognition component;

an application program running under the control of the operating system; and

application program interfaces associated with the handwriting recognition component, said application program interfaces operative to receive data from the application and send data to the application.

2. The computer system of claim 1, wherein the application program interfaces comprise:

a first interface that receives a source handwriting context handle from an application and returns to the application a target handwriting context handle that is based on the source handwriting context handle;

a second interface that receives a first handwriting context handle from an application that causes the handwriting recognition component to destroy the first handwriting context handle;

a third interface that receives from an application an input handwriting context handle and an array of points representing a mouse stroke, and that causes the handwriting recognition component to add the array of points to a data structure represented by the input handwriting context handle;

a fourth interface that receives from an application the input handwriting context handle from an application and that causes the handwriting recognition component to stop adding arrays of points to the data structure represented by the input handwriting context handle;

a fifth interface that receives from an application the input handwriting context handle and that causes the handwriting component to interpret the data structure represented by the input handwriting context handle;

a sixth interface that receives the input handwriting context handle from the application and that returns to the application at least one character that is based on the array of points in the handwriting recognition context; and

a seventh interface that receives the input handwriting context handle and a context character from an application and that causes the handwriting recognition component to interpret the arrays of points based on the context character.

3. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that interfaces with a handwriting recognition component, comprising:

a first interface that receives a source handwriting context handle from an application and returns to the application a target handwriting context handle that is based on the source handwriting context handle;

a second interface that receives a first handwriting context handle from an application that causes the handwriting recognition component to destroy the first handwriting context handle;

a third interface that receives from an application an input handwriting context handle and an array of points representing a mouse stroke, and that causes the handwriting recognition component to add the array of points to a data structure represented by the input handwriting context handle;

a fourth interface that receives from an application the input handwriting context handle from an application and that causes the handwriting recognition component to stop adding arrays of points to the data structure represented by the input handwriting context handle;

a fifth interface that receives from an application the input handwriting

context handle and that causes the handwriting component to interpret the data structure represented by the input handwriting context handle;

a sixth interface that receives the input handwriting context handle from the application and that returns to the application at least one character that is based on the array of points in the handwriting recognition context; and

a seventh interface that receives the input handwriting context handle and a context character from an application and that causes the handwriting recognition component to interpret the arrays of points based on the context character.

4. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, said operating system having a positioning component;

an application program running under the control of the operating system; and

application program interfaces associated with the positioning component, said application program interfaces being functional to allow the application program to cause the positioning component to send and receive data from a positioning device.

5. The computer system of claim 4, wherein the positioning device comprises a Global Positioning System (GPS).

6. The computer system of claim 5, wherein the GPS comprises an Apollo GPS.

7. The computer system of claim 4, wherein the application program interfaces comprise:

a first interface that receives a first device handle from an application, said first device handle referring to the positioning device, and that returns to the application a status value indicating whether or not the positioning device was successfully closed;

a second interface that returns a list of positioning devices to the application; and

a third interface that receives a positioning device profile from an application and that returns to the application a second device handle representing the positioning device, said positioning device being placed in an open state.

8. The computer system of claim 4, wherein the application program interfaces comprise:

a fourth interface that receives a first handle to the positioning device and a first data type from an application and that returns a data value to the application based on the first data type; and

a fifth interface that receives a second handle to the positioning device, a data buffer containing data to be sent to the positioning device, and a second data type from the application and that returns to the application a status indicating whether or not the data buffer was successfully sent to the positioning device.

9. The computer system of claim 8, wherein the first data type is selected from the group consisting of: position, velocity, device state, time, accuracy station, device profile, configuration, settings, differential GPS status, and almanac.

10. The computer system of claim 8, wherein the second data type is selected from the group consisting of: position, velocity, device state, time, accuracy station, device profile, configuration, settings, differential GPS status, and almanac.

11. The computer system of claim 4, wherein the application program interfaces comprise:

a sixth interface that receives a device handle to the positioning device, a data type and a time period from the application, and that causes the positioning component to retrieve data from the positioning device once each time period, said retrieved data based on the data type; and

a seventh interface that receives a second device handle to the positioning device and a data type from an application, and that causes the positioning component to stop retrieving data of the type specified by the data type.

12. The computer system of claim 4, wherein the application program interfaces further comprise an eighth interface the returns to an application the quality of service provided by the positioning device.

13. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that maintains positioning data, comprising:

a first interface that receives a first device handle from an application, said first device handle referring to the positioning device, and that returns to the application a status value indicating whether or not the positioning device was successfully closed;

a second interface that returns a list of positioning devices to the application; and

a third interface that receives a positioning device profile from an application and that returns to the application a second device handle representing the positioning device, said positioning device being placed in an open state.

14. The set of application program interfaces of claim 13, wherein the application program interfaces further comprise:

a fourth interface that receives a first handle to the positioning device and a first data type from an application and that returns a data value to the application based on the first data type; and

a fifth interface that receives a second handle to the positioning device, a data buffer containing data to be sent to the positioning device, and a second data type from the application and that returns to the application a status indicating whether or not the data buffer was successfully sent to the positioning device.

15. The set of application program interfaces of claim 14, wherein the first data type is selected from the group consisting of: position, velocity, device state, time, accuracy station, device profile, configuration, settings, differential GPS status, and almanac.

16. The set of application program interfaces of claim 14, wherein the second data type is selected from the group consisting of: position, velocity, device state, time, accuracy station, device profile, configuration, settings, differential GPS status, and almanac.

17. The set of application program interfaces of claim 13, wherein the application program interfaces further comprise:

a sixth interface that receives a device handle to the positioning device, a data type and a time period from the application, and that causes the positioning

component to retrieve data from the positioning device once each time period, said retrieved data based on the data type; and

a seventh interface that receives a second device handle to the positioning device and a data type from an application, and that causes the positioning component to stop retrieving data of the type specified by the data type.

18. The set of application program interfaces of claim 13, wherein the application program interfaces further comprise an eighth interface the returns to an application the quality of service provided by the positioning device.

19. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a data item indicating a position and a data item indicating a time that the data item indicating a position was set;

a second field comprising almanac data received from a positioning device operably coupled to an embedded system;

a third field comprising an indicator indicating whether the second field is initialized upon startup of the embedded system;

a fourth field comprising an indicator indicating whether the data item indicating a position is initialized upon startup of the embedded system; and

a fifth field comprising an indicator indicating whether the data item indicating a time is initialized upon startup of the embedded system.

20. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a manufacturer name for a positioning device;

a second field comprising a name for the chip manufacturer and chip model of the positioning device;

a third field comprising a number of applications using the positioning device;

a fourth field comprising the quality of data provided by the positioning device;

a fifth field comprising a pointer to a data structure describing the next positioning device; and

a sixth field identifying a communications port used by the positioning device.

21. A computer readable medium having stored thereon a data structure comprising:

a first field comprising the state of a positioning device; and

a second field comprising a time indicating when the first field was updated.

22. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a positioning device mode for a positioning device;

a second field comprising an operational mode for the positioning device;

a third field comprising a correction status for the positioning device;

a fourth field comprising a time indicating when the first field, second field and third field were set; and

a fifth field comprising a maximum age limit assigned to the positioning device.

23. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a station number identifying a station;

a second field indicating whether the station identified by the first field is used during a predetermined data processing step that calculates a position;

a third field comprising an elevation of the station;

a fourth field comprising an azimuth value for the station; and

a fifth field comprising the strength of the signal received from the station.

24. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a position for a positioning device coupled to an embedded system; and

a second field comprising a time when the position of the first field was acquired.

25. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that provides text output, comprising:

a first interface that receives an application identifier, a notification interface, an identifier for the notification interface, a flag identifying a set of notifications to be sent to the notification interface, and a reference to a site information structure and that registers the application with a text-to-speech component; and

a second interface that receives a buffer containing text, a priority flag indicating the type of text, and a buffer that contains text-to-speech control tags and that causes the text-to-speech component to convert the buffer containing text to audio output.

26. The set of application program interfaces of claim 25, further comprising:

a third interface that causes the text-to-speech component to stop playing

the buffer containing text and to flush a set of pending text from a playback queue;

a fourth interface that causes the text-to-speech component to pause playing the buffer containing text; and

a fifth interface that causes the text-to-speech component to resume playing the buffer containing text.

27. The set of application program interfaces of claim 25, further comprising:

a sixth interface that returns a flag indicating the current speech status;

a seventh interface that receives a first talking speed that causes the text-to-speech component to output text at the first talking speed;

an eighth interface that returns a current talking speed;

a ninth interface that receives a first voice identifier that indicates a voice to be used by the text-to-speech component; and

a tenth interface that returns a second voice identifier that indicates the current voice used by the text-to-speech component.

28. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that manages at least one voice command menu, comprising:

a first interface that receives a handle of a window associated with the at least one voice command menu and a flag indicating when the menu should be active in relation to a speech recognition status;

a second interface that receives a list of command structures, each of said command structures describing a voice command, and that returns a number associated with a first voice command added to the at least one voice command menu;

a third interface that deactivates the at least one voice command menu; and

a fourth interface that receives a number corresponding to a first voice command, a number of voice commands to remove and that removes the number of voice commands from the at least one voice command menu, said removal starting with the number corresponding to the first voice command.

29. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that manages a voice command menu, comprising:

a first interface that receives an enablement parameter from an application, said enablement parameter operative to cause a voice recognition component to enable voice recognition when the enablement parameter has a first value and to disable voice recognition when the enablement parameter has a second value; and

a second interface that returns a second parameter to the application, said second parameter operative to indicate that voice recognition is enabled when the second parameter has the first value and that voice recognition is disabled when the second parameter has the second value.

30. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that manages a voice command menu, comprising:

a first interface that receives a first voice command structure identifying a voice menu and a command string, said voice command structure having an association with a second application;

a second interface that receives an identifier of a recognized voice command, a second voice command structure identifying a voice menu associated with the recognized voice command, a verification required flag, an action data string, a list containing at least one recognized phrase of the

recognized voice command, and a command string corresponding the recognized command;

a third interface that is called when a spoken phrase is detected by a voice command component; and

a fourth interface that receives a type of interference detected by the voice command component.

31. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an application that manages a voice command menu, comprising:

a first interface that receives a menu identifier structure, said menu identifier structure comprising an application name and a state name, a language identifier structure and a mode flag from an application that causes a voice recognition system to create a voice command menu identified by the menu identifier structure; and

a second interface that receives the menu identifier structure from an application and that causes the voice recognition system to delete the voice command menu identified by the menu identifier structure.

32. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, said operating system having a speech-to-text component;

an application program running under the control of the operating system;

application program interfaces associated with the speech-to-text component, said application program interfaces operative to receive data from the application and send data to the application.

33. The computer system of claim 32, wherein the application program interfaces comprise:

a first interface that receives an application identifier, a notification interface, an identifier for the notification interface, a flag identifying a set of notifications to be sent to the notification interface, and a reference to a site information structure and that registers the application with a text-to-speech component; and

a second interface that receives a buffer containing text, a priority flag indicating the type of text, and a buffer that contains text-to-speech control tags and that causes the text-to-speech component to convert the buffer containing text to audio output.

34. The computer system of claim 32, wherein the application program interfaces comprise:

a third interface that causes the text-to-speech component to stop playing the buffer containing text and to flush a set of pending text from a playback queue;

a fourth interface that causes the text-to-speech component to pause playing the buffer containing text; and

a fifth interface that causes the text-to-speech component to resume playing the buffer containing text.

35. The computer system of claim 32, wherein the application program interfaces comprise:

a sixth interface that returns a flag indicating the current speech status;

a seventh interface that receives a first talking speed that causes the text-to-speech component to output text at the first talking speed;

an eighth interface that returns a current talking speed;

a ninth interface that receives a first voice identifier that indicates a voice to be used by the text-to-speech component; and

a tenth interface that returns a second voice identifier that indicates the current voice used by the text-to-speech component.

36. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, said operating system having a voice recognition component; and

an application program running under the control of the operating system;

application program interfaces associated with the voice recognition component, said application program interfaces operative to receive data from the application and send data to the application.

37. The computer system of claim 36, wherein the application program interfaces comprise:

a first interface that receives a handle of a window associated with the at least one voice command menu and a flag indicating when the menu should be active in relation to a speech recognition status;

a second interface that receives a list of command structures, each of said command structures describing a voice command, and that returns a number associated with a first voice command added to the at least one voice command menu;

a third interface that deactivates the at least one voice command menu; and

a fourth interface that receives a number corresponding to a first voice command, a number of voice commands to remove and that removes the number

of voice commands from the at least one voice command menu, said removal starting with the number corresponding to the first voice command.

38. The computer system of claim 36, wherein the application program interfaces comprise:

a first interface that receives an enablement parameter from the application, said enablement parameter operative to cause the voice recognition component to enable voice recognition when the enablement parameter has a first value and to disable voice recognition when the enablement parameter has a second value; and

a second interface that returns a second parameter to the application, said second parameter operative to indicate that voice recognition is enabled when the second parameter has the first value and that voice recognition is disabled when the second parameter has the second value.

39. The computer system of claim 36, wherein the application program interfaces comprise:

a first interface that receives from the application a first voice command structure identifying a voice menu and a command string, said voice command structure having an association with a second application;

a second interface that receives an identifier of a recognized voice command, a second voice command structure identifying a voice menu associated with the recognized voice command, a verification required flag, an action data string, a list containing at least one recognized phrase of the recognized voice command, and a command string corresponding the recognized command;

a third interface that is called when a spoken phrase is detected by the voice recognition component; and

a fourth interface that receives a type of interference detected by the voice recognition component.

40. The computer system of claim 36, wherein the application program interfaces comprise:

a first interface that receives a menu identifier structure, said menu identifier structure comprising an application name and a state name, a language identifier structure and a mode flag from an application that causes a voice recognition system to create a voice command menu identified by the menu identifier structure; and

a second interface that receives the menu identifier structure from an application and that causes the voice recognition system to delete the voice command menu identified by the menu identifier structure.

41. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a command string for a voice command;

a second field comprising a flag having values providing information about the voice command;

a third field comprising a command identifier for the voice command;

a fourth field comprising a description of an action performed in response to the voice command; and

a fifth field comprising a category identifier for the voice command.

42. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a recognition threshold for a voice recognition engine;

a second field comprising an identifier for an input audio device supplying input to the voice recognition engine;

a third field comprising a flag indicating whether voice recognition is enabled;

a fourth field comprising the name of a current microphone for the audio input device identified by the second field;

a fifth field comprising the name of a current speaker that is the audio source; and

a sixth field comprisng an identifier for a speech-recognition mode.

43. A computer readable medium having stored thereon a data structure comprising:

a first field comprising an identifier for an input audio device supplying input to a voice recognition engine;

a second field comprising a flag indicating whether voice recognition is enabled; and

a third field comprising a baseline average talking speed for the voice recognition engine.

44. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, said operating system having an out of memory module;

application program interfaces associated with the out of memory module, said application program interfaces being functional to allow the operating system to cause the out of memory module to respond to a low memory condition.

45. The computer system of claim 44, wherein the application program interfaces comprise:

a first interface that receives from the operating system a list of window structures that identify windows to be closed by the out of memory module; and

a second interface called by the out of memory module that causes the operating system to determine if memory is critically low.

46. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an out of memory module of an operating system, comprising:

a first interface that receives from the operating system a list of window structures that identify windows to be closed by the out of memory module; and

a second interface called by the out of memory module that causes the operating system to determine if memory is critically low.

47. A computer readable medium having stored thereon a data structure comprising:

a first field comprising a handle representing a folder containing a local object and a remote object;

a second field comprising a handle representing the local object;

a third field comprising a handle the remote object;

a fourth field comprising a name of the local object;

a fifth field comprising a description of the local object;

a sixth field comprising a name of the remote object; and

a seventh field comprising a description of the remote object; and

wherein during a predetermined data processing operation the fourth, fifth, sixth and seventh fields are displayed.

48. A computer readable medium having stored thereon a data structure comprising:

a first field comprising an object type name;

a second field comprising at least one indicator describing a file system object, said indicators including a changed indicator and a deleted indicator;

a third field comprising an identifier for a file system object;

a fourth field comprising a count of a number of file system object identifiers that are to be replicated if the changed indicator is set, otherwise comprising a count of a number of file system object identifiers in a list of changed objects if both the changed indicator and the deleted indicator are not set; and

a fifth field comprising a count of a number of deleted object identifiers that are to be replicated if the deleted indicator is set, otherwise comprising a count of a number of file object identifiers in a list of unchanged objects if both the changed indicator and the delete indicator are not set.

49 A computer readable medium having stored thereon a data structure comprising:

a first field comprising the name of an object type;

a second field comprising a number of existing objects having the object type named in the first field; and

a third field comprising a timestamp, said timestamp indicating a last time that an object having the object type named in the first field was modified.

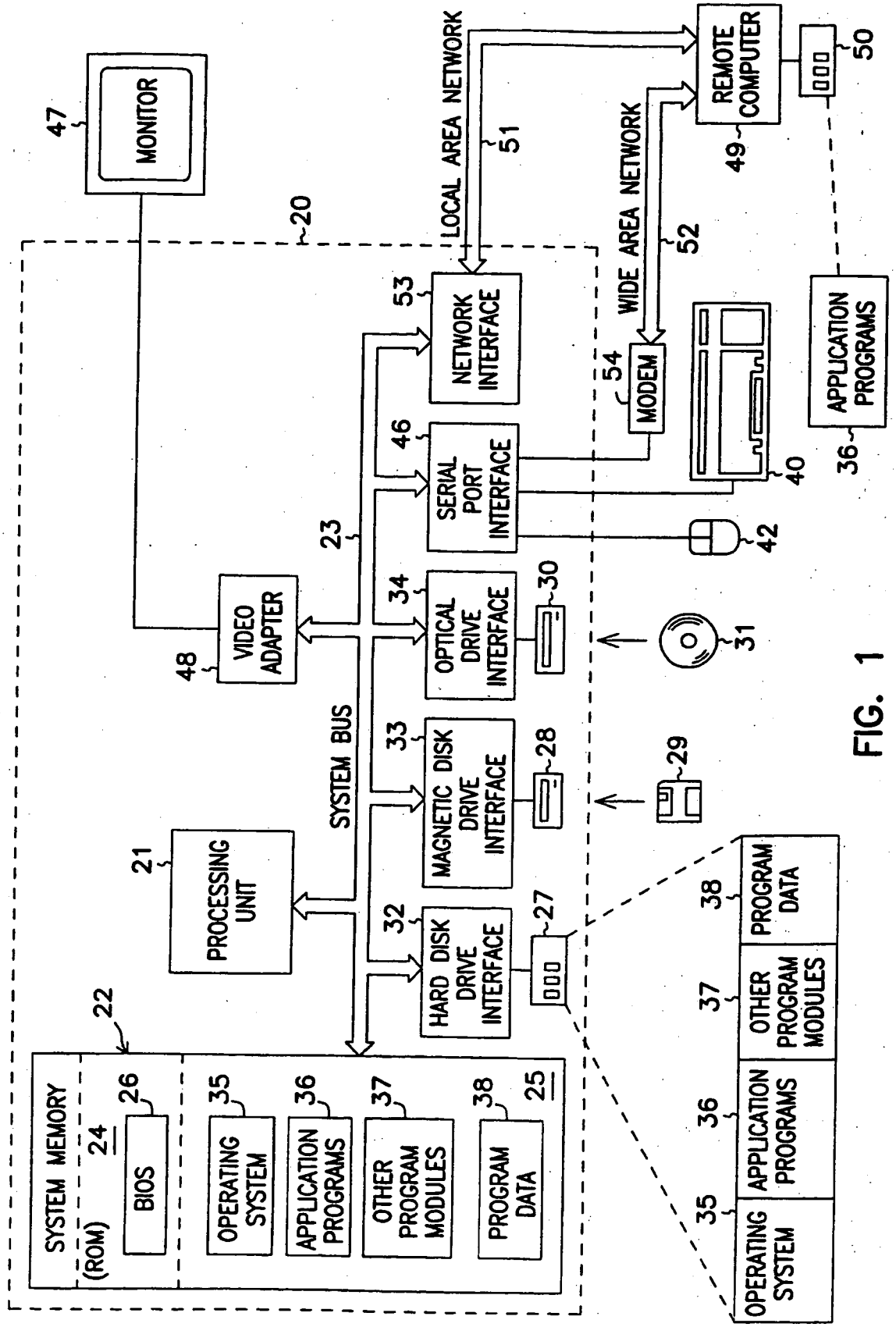

MONITOR
47
20
LOCAL AREA NETWORK
51
REMOTE COMPUTER
49
50
WIDE AREA NETWORK
52
NETWORK INTERFACE
53
MODEM
54
APPLICATION PROGRAMS
36
40
SERIAL PORT INTERFACE
46
42
23
VIDEO ADAPTER
48
OPTICAL DRIVE INTERFACE
34
30
31
SYSTEM BUS
MAGNETIC DISK DRIVE INTERFACE
33
28
29
PROCESSING UNIT
21
HARD DISK DRIVE INTERFACE
32
27
22
SYSTEM MEMORY (ROM)
24
BIOS
26
OPERATING SYSTEM
35
APPLICATION PROGRAMS
36
OTHER PROGRAM MODULES
37
PROGRAM DATA
38
25
PROGRAM DATA
38
OTHER PROGRAM MODULES
37
APPLICATION PROGRAMS
36
OPERATING SYSTEM
35

FIG. 1

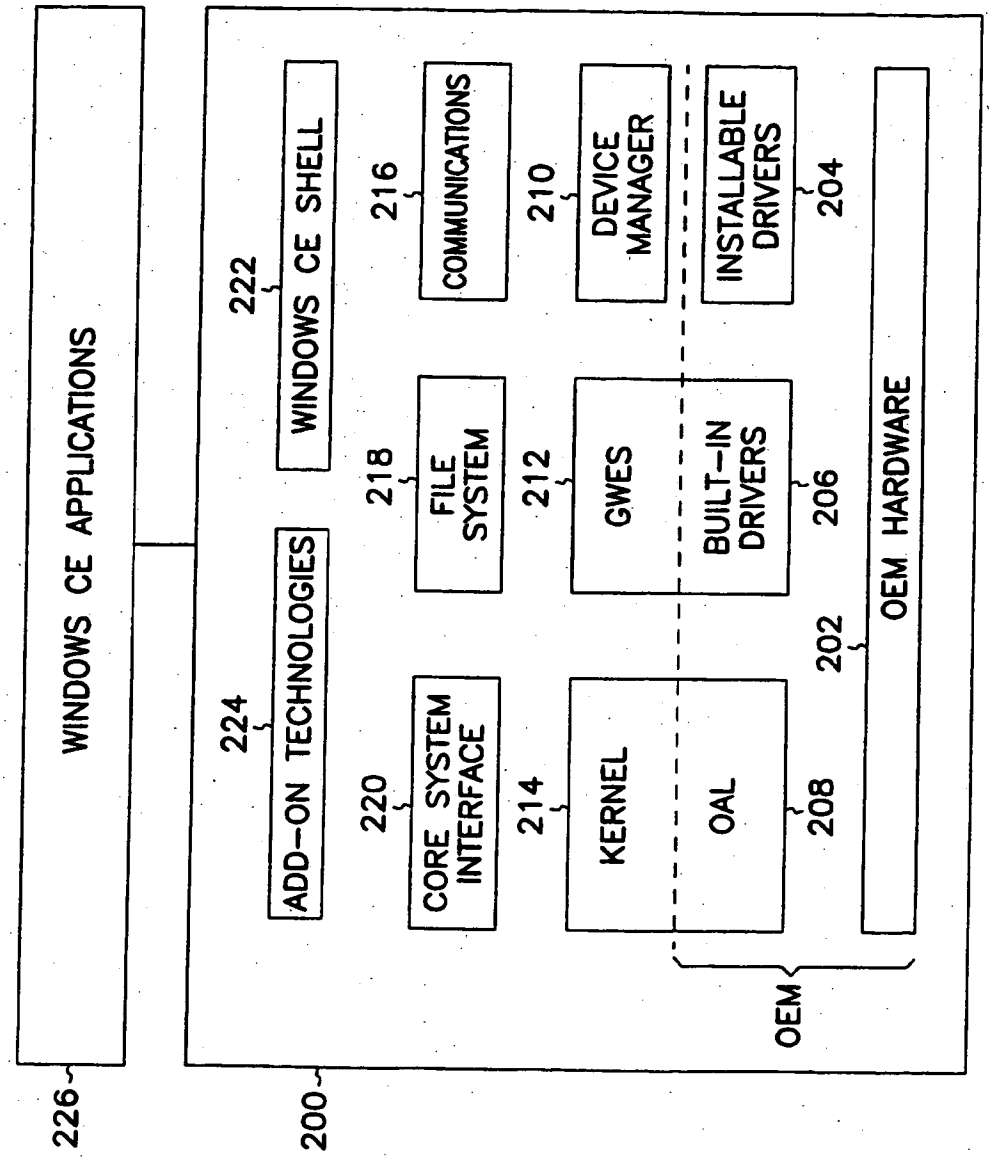

WINDOWS CE APPLICATIONS
226
200
WINDOWS CE SHELL
222
ADD-ON TECHNOLOGIES
224
CORE SYSTEM INTERFACE
220
KERNEL
214
OAL
208
OEM
FILE SYSTEM
218
GWES
212
BUILT-IN DRIVERS
206
COMMUNICATIONS
216
DEVICE MANAGER
210
INSTALLABLE DRIVERS
204
OEM HARDWARE
202

FIG. 2

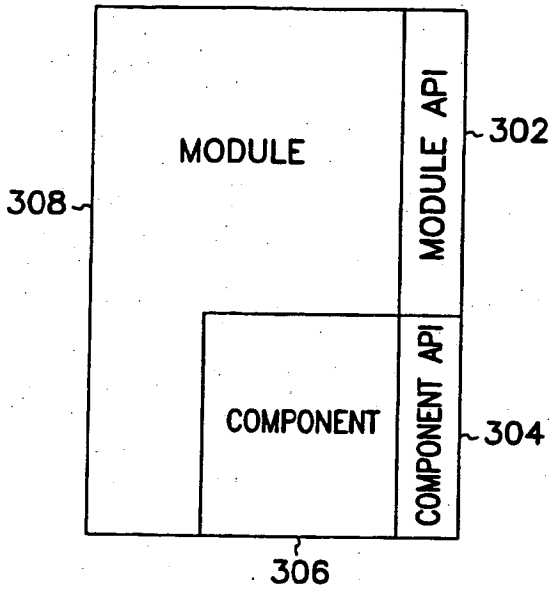


FIG. 3

INTERNATIONAL SEARCH REPORT

Int ional Application No
PCT/US 99/06223

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category '	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 724 506 A (CLARON ET AL) 3 March 1998 (1998-03-03) the whole document ---	1-49
Y	LEVY M: "WINDOWS CE AT THE CENTER OF A JUGGLING ACT" EDN ELECTRICAL DESIGN NEWS, vol. 42, no. 15, 17 July 1997 (1997-07-17), pages 38, 40, 42, 44,-46,48,50, XP000754502 Newton, MA, US ISSN: 0012-7515 page 40, left-hand column, line 41 - line 51 page 40, middle column, line 44 - line 54 --- -/--	1-49

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

' Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 July 1999	04/08/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Fonderson, A

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Ir. ational Application No
PCT/US ~~99/06223~~

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MENDELSOHN N: "Operating systems for component software environments" PROCEEDINGS. THE SIXTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS (CAT. NO.97TB100133), PROCEEDINGS. THE SIXTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS (CAT. NO.97TB100133), CAPE COD, MA, USA, 5-6 MAY 1997, pages 49-54, XP002109963 1997, Los Alamitos, CA, USA, IEEE Comput. Soc. Press, USA. ISBN: 0-8186-7834-8 the whole document ---	1-49
A	BRIAN N. BERSHAD ET AL.: "Extensibility, Safety and Performance in the SPIN Operating System" OPERATING SYSTEMS REVIEW (SIGOPS)., vol. 29, no. 5, December 1995 (1995-12), pages 267-284, XP002109964 NEW YORK, US the whole document -----	1-49

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/US 99/06223

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5724506 A	03-03-1998	AU 5853696 A	29-11-1996
		EP 0769169 A	23-04-1997
		WO 9635991 A	14-11-1996